Formal Models and Techniques for Analyzing Security Protocols: A Tutorial

Véronique Cortier, Steve Kremer

Foundations and Trends® in Programming Languages
Vol. 1, No. 3 (2014) 151–267

DOI: 10.1561/2500000001

now
the essence of knowledge

Formal Models and Techniques for Analyzing Security Protocols: A Tutorial

Véronique Cortier
LORIA, CNRS
cortier@loria.fr

Steve Kremer
INRIA Nancy
steve.kremer@inria.fr

Contents

Abstract

Security protocols are distributed programs that aim at securing communications by the means of cryptography. They are for instance used to secure electronic payments, home banking and more recently electronic elections. Given the financial and societal impact in case of failure, and the long history of design flaws in such protocols, formal verification is a necessity. A major difference from other safety critical systems is that the properties of security protocols must hold in the presence of an arbitrary adversary. The aim of this paper is to provide a tutorial to some modern approaches for formally modeling protocols, their goals and automatically verifying them.

V. Cortier and S. Kremer. *Formal Models and Techniques for Analyzing Security Protocols: A Tutorial*. Foundations and Trends® in Programming Languages, vol. 1, no. 3, pp. 151–267, 2014.
DOI: 10.1561/2500000001.

1

Introduction

Security protocols are used to protect electronic transactions. The probably most used security protocol is the SSL/TLS protocol which underlies the https protocol in web browsers. It may be used for electronic commerce, or simply to encrypt web search queries on their way between the host and the search engine. There are of course many other protocols in use, e.g. to authenticate to providers on mobile phones or withdraw cash on an ATM. Moreover, the digitalization of our modern society requires the use of security protocols in an increasing number of contexts, such as electronic passports that may include RFID chips, electronic elections that allow for Internet voting, etc.

We may think of security protocols as distributed programs that make use of cryptography, e.g. encryption, to achieve a security property, such as confidentiality of some data, e.g. your credit card number. Given the difficulty of designing correct distributed systems in general, it is not surprising that many flaws were discovered in security protocols, even without breaking the underlying cryptography. During the last 30 years many research efforts were spent on designing techniques and tools to analyze security protocols. One may trace this line of work back to the seminal work of Dolev and Yao [1981] who

pioneered the ideas of an attacker who completely controls the communication network, has an unbounded computational power, but manipulates protocol messages according to some predefined rules, idealizing the protections offered by cryptography. These techniques not only allowed to better understand the principles underlying secure protocol design, but also resulted in mature tools, for automated protocol analysis, and the discovery of many attacks. For example, while designing a formal model of Google's Single Sign-On protocol, that allows a user to identify himself only once and then access various applications (such as Gmail or Google calendar), Armando et al. [2008] discovered that a dishonest service provider could impersonate any of its users at another service provider. This flaw has been corrected since. Basin et al. [2012] have identified flaws and proposed fixes for the ISO/IEC 9798 standard for entity authentication, using automated protocol verification tools. The standard has been revised to include their proposed amendments. Bortolozzo et al. [2010] designed a dedicated analysis tool for hardware security tokens that implement the PKCS#11 standard. The tool automatically reverse-engineers the tokens to extract its configuration, builds an abstract model to be analyzed and verifies the attack on the token if an attack is found. They were able to find unknown attacks on more than 10 commercial tokens.

This paper proposes a tutorial, presenting modern techniques to model and automatically analyze security protocols. Given the large body of work in this area we do not aim to be exhaustive and only present some selected methods and results. We expect that this tutorial could serve as a basis for a master, or graduate course, or allow researchers from different areas to get an overview of the kinds of techniques that are used. The outline of the tutorial is as follows.

- We first present an informal description of our running example, the Needham Schroeder public key protocol that we used for illustration purposes in the remainder of the paper.
- Then, we explain how protocol messages can be modeled as first order terms, and how adversary capabilities can be modeled by an inference system. We also provide a decision algorithm for deduction, i.e. the adversary's capability to construct new messages.

- Next, we introduce a more general model, based on equational theories. We revisit deduction and define a notion of message indistinguishability, called static equivalence. We again provide a decision procedure for static equivalence for a simple equational theory representing symmetric encryption.

- We continue by introducing a process calculus, the applied pi calculus, which we use to model protocols. One of the main differences with the original pi calculus is that the calculus allows communication of messages represented by terms, rather than only names. We illustrate how protocols can be conveniently modeled in this formalism.

- Next we discuss how we can express security properties of protocols modelled in the applied pi calculus. We cover different flavors of confidentiality, authentication, but also anonymity properties, expressed as behavioral equivalences of processes.

- We go on discussing automated verification. We first consider the case when protocol participants only execute a bounded number of sessions. We present a decision procedure based on constraint solving which allows to decide secrecy in this setting.

- Finally, we show that the general case, where the number of sessions is unbounded, is undecidable. We show that nevertheless it is possible to design tools that are able to analyze protocols. This comes at the cost that termination is not guaranteed. In particular, we present an approach based on a representation of the protocol and the adversary as Horn clauses and describe a resolution based procedure implemented in the ProVerif tool.

- We conclude the tutorial by briefly discussing some other approaches for automated verification and other directions in this research area.

2

Running example

We first introduce our running example, the Needham Schroeder public key protocol [Needham and Schroeder, 1978]. We will also describe the famous man in the middle attack, discovered by Lowe [1996] 17 years after the publication of the original paper.

2.1 The Needham Schroeder public key protocol

The Needham Schroeder public key protocol can be described by the following three message transmissions.

$$\begin{array}{lll} 1. & A \to B: & \{\!|\langle A, N_a\rangle|\!\}^a_{\mathsf{pk}(B)} \\ 2. & B \to A: & \{\!|\langle N_a, N_b\rangle|\!\}^a_{\mathsf{pk}(A)} \\ 3. & A \to B: & \{\!|N_b|\!\}^a_{\mathsf{pk}(B)} \end{array}$$

The notation $A \to B:\ m$ denotes that Alice (A) is sending the message m to Bob (B). We use the notation $\{\!|m|\!\}^a_{\mathsf{pk}(B)}$ to denote the asymmetric encryption of the message m with B's public key. $\langle m_1, m_2\rangle$ denotes the pairing, i.e., the concatenation, of messages m_1 and m_2.

In the first message Alice sends her identity A and a nonce N_a encrypted with Bob's public key to Bob. A nonce, (which stands for

number used once) is a fresh number that is randomly generated in each session.

When Bob receives this first message, he decrypts the message and checks that it is well-formed. Bob also generates a nonce N_b and encrypts the nonces N_A and N_B with Alice's public key. This mechanism of sending someone an encrypted nonce and waiting for the recipient to send back this nonce is often called a *challenge-response* mechanism. Here Alice's *challenge* to Bob is to be able to decrypt the message and Bob's response consists in sending back the nonce.

When receiving the second message Alice decrypts it and verifies that it contains her previous nonce N_a. This proves to Alice that Bob has indeed received the first message. Indeed, Bob is the only entity able to decrypt the first message, as it was encrypted with his public key $\mathsf{pk}(B)$. Then Alice retrieves the second nonce N_b and replies with the encryption of N_b with Bob's public key. Bob can decrypt this message and verify that it contains the nonce N_b generated previously. We can observe that the second and third message also form a challenge-response from Bob to Alice.

The aim of the protocol is to guarantee *mutual authentication*: at the end of a successful protocol run Bob should be ensured he has been communicating with Alice while Alice has been communicating with Bob. Moreover, the protocol should guarantee *confidentiality* of the nonces N_a and N_b.

While the above presentation of the protocol is convenient to read it does however lack a formal semantics and many details are left implicit, e.g. which messages are freshly generated, etc. It also only shows one possible execution of the protocol, which is the honest execution where everyone behaves as intended. However the network used for communication is not reliable, i.e. an attacker may intercept messages, re-route and modify them. Moreover, this notation leaves implicit which part of a message needs to be verified as being well formed, or correspond to some known value.

A slightly more faithful representation of this protocol is given in Figure 2.1. In this figure it is explicit that the messages sent by A are not necessarily received by B. The empty space between outgoing

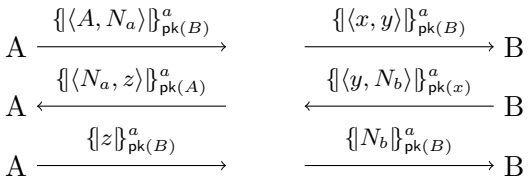

Figure 2.1: Lowe's attack on the Needham Schroeder public key protocol.

and incoming arrows represents the untrusted network. We also note the use of variables in some messages. Variables are used whenever the value of part of the message is a priori unknown. These variables may then be used to construct other messages later. For example, the second message from A's point of view $\{\!|\langle N_a, z\rangle|\!\}^a_{\mathsf{pk}(A)}$ explicits that A needs to check that the first component of the encrypted pair corresponds to the nonce N_a while the second component is unknown and will be bound to variable z. In the third message A uses variable z as the value to be encrypted. This representation is very useful when thinking about *what could go wrong?* in the protocol.

2.2 Lowe's man in the middle attack

Lowe [1996] discovered an attack in the case where A initiates a session with a dishonest party C: C may fool B, by making B believe he is A. Moreover, the nonce N_b which B believes to only share with A becomes known to C. The attack is displayed in Figure 2.2. We write $C(A)$ whenever C is impersonating A. The first message from A to C is decrypted and re-encrypted with B's public key. As the second message is encrypted with A's public key, C cannot modify this message, and forwards it to A. As A has no means to know that N_b was generated by B rather than C, he accepts this message and returns $\{\!|N_b|\!\}^a_{\mathsf{pk}(C)}$. One may say that A acts as a *decryption oracle*. The attacker may now learn N_b and successfully complete the protocol with B.

This execution violates the secrecy of N_b and authentication from B's point of view. B believes he successfully finished a session of the protocol with A while A did not authenticate to B.

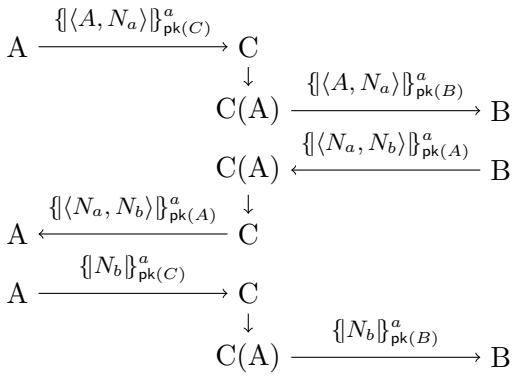

Figure 2.2: Lowe's attack on the Needham Schroeder public key protocol.

The crucial observation is that A is willing to start a session with a corrupted agent. This is actually in contradiction with the assumptions made by Needham and Schroeder [1978]:

> *We also assume that each principal has a secure environment in which to compute, such as is provided by a personal computer or would be by a secure shared operating system. Our viewpoint throughout is to provide authentication services to principals that choose to communicate securely. We have not considered the extra problems encountered when trying to force all communication to be performed in a secure fashion or when trying to prevent communication between particular principals in order to enforce restrictions on information flow.*

It may therefore be debatable whether Lowe's finding should be considered as an attack. Nevertheless, while the above assumption may have been reasonable in 1978, it is certainly not the case anymore. Computers may run malware and people may connect to a malicious server, e.g., because of a phishing attack.

Lowe also shows that the protocol can easily be fixed to avoid this attack by adding B's identity to the second message, i.e. replace the message $\{\!|\langle N_a, N_b\rangle|\!\}^a_{\mathsf{pk}(A)}$ by $\{\!|\langle N_a, \langle N_b, B\rangle\rangle|\!\}^a_{\mathsf{pk}(A)}$. Indeed, this altered

message prevents the man in the middle attack as A would expect the second message to be $\{\!|\langle N_a, \langle N_b, C\rangle\rangle|\!\}^a_{\mathsf{pk}(A)}$ while the intruder can only produce the message $\{\!|\langle N_a, \langle N_b, B\rangle\rangle|\!\}^a_{\mathsf{pk}(A)}$. We will refer to the fixed protocol as the Needham Schroeder Lowe (NSL) protocol.

3

Messages and deduction

Many different symbolic models are used to represent and reason about protocols. Examples of symbolic models are process algebra (e.g. CSP [Schneider, 1997], applied-pi calculus [Abadi and Fournet, 2001]), strand spaces [Thayer et al., 1999], constraint systems [Millen and Shmatikov, 2001, Comon-Lundh and Shmatikov, 2003], or Horn clauses [Blanchet, 2001]. These models have many differences but they all have in common the fact that messages are represented by *terms*. Intuitively, the exact values of keys, identities, or nonces are abstracted away but the structure of the message is kept and modeled as a special labelled graph, called term.

3.1 Terms

Terms are a common object in computer science. We introduce here only the basic definitions of terms, substitutions, and unification and we refer the reader to handbooks, e.g. [Baader and Nipkow, 1998, Baader and Snyder, 2001], for a deeper introduction on terms.

3.1.1 Terms

Cryptographic primitives such as encryption or signatures are simply represented by function symbols. A finite set of function symbols is called a *signature*, where a function symbol f has an associated integer, its arity.

Definition 3.1. Given a set $\mathcal{X}$ of variables and a set $\mathcal{N}$ of names (used to represent atomic data such as keys, nonces, or identities), the set of *terms* of the signature $\mathcal{F}$, the variables $\mathcal{X}$ and the names $\mathcal{N}$ is denoted $T(\mathcal{F}, \mathcal{X}, \mathcal{N})$ and is inductively defined as names, variables, and function symbols applied to other terms.

Variables are typically used to represent parts of messages that are left unspecified, such as components received from the outside. The set of variables occurring in a term t is denoted $\mathsf{var}(t)$, while the set of names of t is denoted $\mathsf{n}(t)$. Terms without variables are called *ground* or *closed*.

Example 3.1. In the context of security protocols, a standard signature is $\mathcal{F}_{\mathsf{std}} = \{\mathsf{senc}, \mathsf{aenc}, \mathsf{pair}, \mathsf{pk}\}$ where senc, aenc and pair are three symbols of arity 2, representing respectively symmetric encryption, asymmetric encryption, and concatenation, while pk is a symbol of arity 1, representing the public key associated to some private key. For example, the term $t_0 = \mathsf{aenc}(\mathsf{pair}(a, n_a), \mathsf{pk}(k_a))$, where $a, n_a, k_a \in \mathcal{N}$, represents the encryption under the public key $\mathsf{pk}(k_a)$ of the concatenation of the identity a together with the nonce n_a. This term t_0 can be used to represent the first message sent by a in the Needham-Schroeder protocol.

For readability, we may write $\langle t_1, t_2 \rangle$ instead of $\mathsf{pair}(t_1, t_2)$. We may also write $\{\!|t_1|\!\}^s_{t_2}$ instead of $\mathsf{senc}(t_1, t_2)$, and $\{\!|t_1|\!\}^a_{t_2}$ for $\mathsf{aenc}(t_1, t_2)$.

The set of *positions* of a term t is written $\mathsf{pos}(t) \subseteq \mathbb{N}^*$. We use ϵ to denote the root position. Formally, the set $\mathsf{pos}(t)$ is inductively defined as follows.

$$\mathsf{pos}(f(t_1, \ldots, t_n)) = \{\epsilon\} \cup \bigcup_{i=1}^{n} i \cdot \mathsf{pos}(t_i)$$

The set of *subterms* of t is written $\mathsf{st}(t)$. The subterm of t at position $p \in \mathsf{pos}(t)$ is written $t|_p$. In particular, $t|_\epsilon = t$. The term obtained by

replacing $t|_p$ with a term u in t is denoted $t[u]_p$. The set of subterms of a set of terms S is simply $\mathsf{st}(S) = \bigcup_{t \in S} \mathsf{st}(t)$.

Example 3.2. Let $t_0 = \mathsf{aenc}(\mathsf{pair}(a, n_a), \mathsf{pk}(k_a))$ as defined in Example 3.1. The set of its positions is $\mathsf{pos}(t_0) = \{\epsilon, 1, 1.1, 1.2, 2, 2.1\}$. The set of its subterms is $\mathsf{st}(t_0) = \{t_0, \mathsf{pair}(a, n_a), a, n_a, \mathsf{pk}(k_a), k_a, \}$ and $t_0[\mathsf{aenc}(n_a, \mathsf{pk}(k_a))]_1 = \mathsf{aenc}(\mathsf{aenc}(n_a, \mathsf{pk}(k_a)), \mathsf{pk}(k_a))$.

3.1.2 Substitutions

Terms with variables are used to represent partially specified messages. For example, in the Needham-Schroeder protocol, the agent B expects an encrypted message that contains A's identity and some unknown part X, that should be the nonce sent by A. However, B cannot control that X is indeed the nonce sent by A. The message expected by B will therefore be represented by the term $\mathsf{aenc}(\mathsf{pair}(a, x), \mathsf{pk}(k_a))$ where x is a variable. Such variables can then be *instantiated* depending on the actually received message, that is, they are replaced by a term. This is formally defined through substitutions.

Definition 3.2 (Substitution)**.** A *substitution* is a function σ from a finite subset, called *domain* and denoted $\mathsf{Dom}(\sigma)$ of the set of variables $\mathcal{X}$ to $T(\mathcal{F}, \mathcal{X}, \mathcal{N})$. When applied to a term, a substitution replaces any variable x by the corresponding term $\sigma(x)$. Formally:

$$\begin{array}{rcll} \sigma(x) & = & x & \text{if } x \notin \mathsf{Dom}(\sigma) \\ \sigma(f(t_1, \ldots, t_n)) & = & f(\sigma(t_1), \ldots, \sigma(t_n)) & \end{array}$$

We often write $t\sigma$ instead of $\sigma(t)$.

The question of whether two terms with variables may be made equal is called *unification*. Two terms u and v are said *unifiable* if there exists a substitution σ, called *unifier*, such that $u\sigma = v\sigma$.

Proposition 3.1 ([Baader and Snyder, 2001])**.** If two terms u and v are unifiable then there exists a *most general unifier* $\mathsf{mgu}(u, v)$ such that any unifier is actually an instance of $\mathsf{mgu}(u, v)$, that is for any σ such that $u\sigma = v\sigma$, there exists θ such that $\sigma = \mathsf{mgu}(u, v)\theta$.

3.2 Message deduction

Cryptographic primitives have of course particular properties. For example, anyone can decrypt a message with the corresponding decryption key. These properties are reflected through inference rules, that define which messages can be computed from an a priori given set of messages.

3.2.1 Inference system

From the key k and the message $\mathsf{senc}(m, k)$, which represents the (symmetric) encryption of m over k, one can compute m. This can be represented by the rule

$$\frac{\mathsf{senc}(m,k) \quad k}{m}$$

This can be formalized as follows.

Definition 3.3. An *inference rule* is a rule of the form: $\dfrac{u_1 \quad \cdots \quad u_n}{u}$ with $u_1, \ldots, u_n, u$ are terms (with variables).

An *inference system* is a set of inference rules.

The standard inference system corresponding to the encryption and concatenation is presented in Figure 3.1. It is often called the Dolev-Yao system, in reference to the first symbolic formalisation of the deductive power of an attacker by Dolev and Yao [1981]. The first line of Figure 3.1 corresponds to the fact that anyone can concatenate terms and retrieve terms from a concatenation. The second line models that one can encrypt and decrypt symmetrically whenever he/she has the corresponding key. Similarly, the third line corresponds to asymmetric encryption. The first inference rule of each line of Figure 3.1 is called a *composition rule* while the other rules are called *decomposition rules*.

Of course, more inference rules may be considered, for example to represent signatures, hashes, etc. Other inference systems are presented in Exercises (§3.4).

$$\mathcal{I}_{\mathsf{DY}} : \left\{ \begin{array}{ll} \dfrac{x \quad y}{\langle x, y\rangle} & \dfrac{\langle x, y\rangle}{x} \quad \dfrac{\langle x, y\rangle}{y} \\[2ex] \dfrac{x \quad y}{\mathsf{senc}(x, y)} & \dfrac{\mathsf{senc}(x, y) \quad y}{x} \\[2ex] \dfrac{x \quad y}{\mathsf{aenc}(x, y)} & \dfrac{\mathsf{aenc}(x, \mathsf{pk}(y)) \quad y}{x} \end{array} \right.$$

Figure 3.1: Inference system $\mathcal{I}_{DY}$ corresponding to a "Dolev-Yao adversary".

3.2.2 Derivability

Inference rules may be combined to compute or *derive* new messages. For example, from a set of messages $S_0 = \{\langle k_1, k_2\rangle, \langle k_3, a\rangle, \{\!|n|\!\}^s_{\langle k_1, k_3\rangle}\}$, an attacker may learn k_1 from $\langle k_1, k_2\rangle$ and k_3 from $\langle k_3, a\rangle$. This allows him to obtain $\langle k_1, k_3\rangle$, which in turn enables him to decrypt $\{\!|n|\!\}^s_{\langle k_1, k_3\rangle}$ to get n. He may then further append a to it for example, yielding $\langle n, a\rangle$. We say that $\langle n, a\rangle$ is *deducible* from S_0 and the corresponding deduction steps can be conveniently represented by the following proof tree.

$$\dfrac{\dfrac{\dfrac{\langle k_1, k_2\rangle}{k_1} \quad \dfrac{\langle k_3, a\rangle}{k_3}}{\langle k_1, k_3\rangle}}{\dfrac{\{\!|n|\!\}^s_{\langle k_1, k_3\rangle} \quad \langle k_1, k_3\rangle}{n} \quad \dfrac{\langle k_3, a\rangle}{a}}{\langle n, a\rangle}$$

Formally, the notion of *deduction* is defined as follows.

Definition 3.4 (deduction)**.** Let $\mathcal{I}$ be an inference system. A term t is *deducible in one step* from a set of terms S for the inference system $\mathcal{I}$, which is denoted $S \vdash^1_{\mathcal{I}} t$ if there exists an inference rule $\dfrac{u_1 \quad \cdots \quad u_n}{u}$ of $\mathcal{I}$, terms $t_1, \ldots, t_n \in S$, and a substitution θ such that

$$t_i = u_i\theta \quad \text{for} \quad 1 \leq i \leq n \quad \text{and} \quad t = u\theta$$

We say that $\dfrac{t_1 \quad \cdots \quad t_n}{t}$ is an *instance* of $\dfrac{u_1 \quad \cdots \quad u_n}{u}$.

A term t is *deducible* from a set of terms S, denoted $S \vdash_{\mathcal{I}} t$ if there exists a *proof tree* Π, that is, a tree such that

- the leaves of Π are labelled by terms in S;
- if a node of Π is labelled by t_{n+1} and has n child nodes labelled by $t_1, \ldots, t_n$ respectively then $\dfrac{t_1 \quad \cdots \quad t_n}{t_{n+1}}$ is an instance of some inference rule of $\mathcal{I}$;
- the root of Π is labelled by t.

The set $\mathsf{Terms}(\Pi)$ is the set of all terms that appear in any node of Π. We may write $S \vdash t$ instead of $S \vdash_{\mathcal{I}} t$ when $\mathcal{I}$ is clear from the context.

Example 3.3. Let $S_0 = \{\langle k_1, k_2\rangle, \langle k_3, a\rangle, \{\!|n|\!\}^s_{\langle k_1,k_3\rangle}\}$ and $\mathcal{I}_{DY}$ as defined in Figure 3.1. The fact that $S_0 \vdash \langle k_1, k_2\rangle$ is witnessed by the proof tree consisting of a single node labelled $\langle k_1, k_2\rangle$. Then k_1 and k_3 are derivable in one step from S_0. We have $S_0 \vdash \langle n, a\rangle$ and a proof tree of it is the tree presented in Section 3.2.2.

3.3 An algorithm to decide message deduction

To analyse security protocols, one of the key methods is to design automatic procedures to detect whether a protocol is subject to attacks. In this context, a very basic question is to decide whether a term t is deducible from a (finite) set of terms S. This problem is called the *intruder deduction problem*.

Definition 3.5 (Intruder deduction problem)**.** Let $\mathcal{I}$ be an inference system. The intruder deduction problem consists in deciding the following problem:

Input a finite set of terms S and a term t

Output whether $S \vdash_{\mathcal{I}} t$.

The intruder deduction problem is undecidable in general [Abadi and Cortier, 2006], for arbitrary inference systems. It is however easily decidable for the case of *local theories* that satisfy that whenever $S \vdash t$ then there exists a proof of it that uses only subterms of S and t.

Definition 3.6. An inference system $\mathcal{I}$ is *local* if for any finite set of terms S and for any term t such that $S \vdash t$ there exists a proof tree Π of $S \vdash t$ such that any term labeling Π is in $\mathsf{st}(S \cup \{t\})$.

Theorem 3.1. Let $\mathcal{I}$ be a local inference system. The intruder deduction system is decidable in polynomial time (PTIME).

Proof. Let $\mathcal{I}$ be a local inference system. Whether $S \vdash t$ can be decided by the following algorithm:

- Let $S_0 = S$.
- Let $S_{i+1} = S_i \cup (\{u \mid S_i \vdash^1 u\} \cap \mathsf{st}(S \cup \{t\}))$.
- If $S_{i+1} = S_i$ then stop.
- Check whether $t \in S_i$.

Let N be the cardinality of $\mathsf{st}(S \cup \{t\})$. For any i, we have that $S_i \subseteq \mathsf{st}(S \cup \{t\})$ therefore the procedure stops in at most N steps. Since each step can be computed in polynomial time, the overall procedure is polynomial. Its correctness is ensured by the locality property of $\mathcal{I}$. □

The intruder deduction problem is decidable in PTIME for the Dolev-Yao inference system, due to its locality property.

Proposition 3.2. The Dolev-Yao inference system $\mathcal{I}_{DY}$ is local.

Proof. We say that a proof tree Π of $S \vdash t$ is *minimal* if its number of nodes is minimal.

We show a property that is slightly stronger than locality:
For any S, t such that $S \vdash t$, for any minimal proof Π of $S \vdash t$, it holds that $\mathsf{Terms}(\Pi) \subseteq \mathsf{st}(S \cup \{t\})$. Moreover if Π is reduced to a leaf or ends with a decomposition rule then $\mathsf{Terms}(\Pi) \subseteq \mathsf{st}(S)$.

The proof proceeds by induction on the size of Π.

Base case. If Π is a leaf then by definition $\mathsf{Terms}(\Pi) \subseteq \mathsf{st}(S)$.

Induction. Let Π be a minimal proof of $S \vdash t$ and consider the last applied inference rule. Assume first it is a composition rule:

$$\Pi = \left\{ \dfrac{\dfrac{\Pi_1}{t_1} \qquad \dfrac{\Pi_2}{t_2}}{t} \right.$$

with $t = f(t_1, t_2)$ for $f \in \{\mathsf{pair}, \mathsf{aenc}, \mathsf{senc}\}$. By induction hypothesis, it must be the case that $\mathsf{Terms}(\Pi_1) \subseteq \mathsf{st}(S \cup \{t_1\}) \subseteq \mathsf{st}(S \cup \{t\})$ and $\mathsf{Terms}(\Pi_2) \subseteq \mathsf{st}(S \cup \{t_2\}) \subseteq \mathsf{st}(S \cup \{t\})$. Hence $\mathsf{Terms}(\Pi) \subseteq \mathsf{st}(S \cup \{t\})$.

Consider now the case where the last applied inference rule is a decomposition and assume first that it corresponds to asymmetric decryption, that is $\Pi = \left\{ \dfrac{\dfrac{\Pi_1}{\mathsf{aenc}(t, \mathsf{pk}(t_2))} \qquad \dfrac{\Pi_2}{t_2}}{t} \right.$. The last applied rule in Π_1 must be a decomposition. Indeed, if it were a composition, we would have $\Pi = \left\{ \dfrac{\dfrac{\dfrac{\Pi_1'}{t} \qquad \dfrac{\Pi_1''}{\mathsf{pk}(t_2)}}{\mathsf{aenc}(t, \mathsf{pk}(t_2))} \qquad \dfrac{\Pi_2}{t_2}}{t} \right.$. Therefore Π_1' would be a smaller proof of $S \vdash t$, which contradicts the minimality of Π. Now, since the last applied rule of Π_1' is a decomposition and applying the induction hypothesis, we deduce that $\mathsf{Terms}(\Pi_1) \cup \{\mathsf{aenc}(t, \mathsf{pk}(t_2))\} \subseteq \mathsf{st}(S)$ and $\mathsf{Terms}(\Pi_2) \subseteq \mathsf{st}(S \cup \{t_2\})$. We deduce that $\mathsf{aenc}(t, \mathsf{pk}(t_2))$ and therefore t and t_2 are in $\mathsf{st}(S)$. Hence $\mathsf{Terms}(\Pi) \subseteq \mathsf{st}(S \cup \{t\})$.

The other cases are left to the reader. □

Corollary 3.2. The intruder deduction problem is decidable in PTIME for the Dolev-Yao inference system $\mathcal{I}_{DY}$.

This follows directly from Theorem 3.1 and Proposition 3.2.

3.4 Exercises

Exercise 1 (*). We consider the inference system $\mathcal{I}_{\mathsf{DY}}$ introduced in Section 3.2.2. Let

$$S = \{\{k_2\}_{\langle k_1, \{k_1\}_{k_3}\rangle}, \langle k_1, k_1\rangle, \{\{k_1\}_{k_3}\}_{k_1}\}.$$

1. Show that k_1 and k_2 are deducible from S, that is $S \vdash_{\mathcal{I}_{\mathsf{DY}}} k_1$ and $S \vdash_{\mathcal{I}_{\mathsf{DY}}} k_2$.

2. Show that k_3 is not deducible from S, that is $S \nvdash_{\mathcal{I}_{\mathsf{DY}}} k_3$.

Exercise 2 (**). We consider two binary symbols sign and blind. Intuitively, the term $\mathsf{sign}(m, k)$ is meant to represent the message m signed by the key k while $\mathsf{blind}(m, r)$ represents the message m *blinded* by the random factor r. *Blind signatures* have the following property: given the signature of a blinded message and given the blinding factor, one can compute a valid signature of the original message. This is reflected by the following deduction rule:

$$\frac{\mathsf{sign}(\mathsf{blind}(m, r), k) \quad r}{\mathsf{sign}(m, k)}$$

Such blind signatures are used for example in voting protocols [Fujioka et al., 1992].

The complete inference system corresponding to this primitive is $\mathcal{I}_{\mathsf{sign}}$ defined as follows.

$$\mathcal{I}_{\mathsf{sign}} : \left\{ \begin{array}{llll} & \dfrac{x \quad y}{\mathsf{pair}(x,y)} & \dfrac{x \quad y}{\mathsf{blind}(x,y)} & \dfrac{x \quad y}{\mathsf{sign}(x,y)} \\ \dfrac{\mathsf{pair}(x,y)}{x} & \dfrac{\mathsf{pair}(x,y)}{y} & \dfrac{\mathsf{sign}(\mathsf{blind}(x,y),z) \quad y}{\mathsf{sign}(x,z)} & \dfrac{\mathsf{blind}(x,y) \quad y}{x} \end{array} \right.$$

1. Exhibit a set of messages S and a name n such that $S \vdash_{\mathcal{I}_{\mathsf{sign}}} n$ but $S \nvdash_{\mathcal{I}_{\mathsf{DY}}} n$ where $\mathcal{I}_{\mathsf{DY}}$ has been defined in Section 3.2.2.

2. Show that $\mathcal{I}_{\mathsf{sign}}$ is not a local inference system.

3. Provide an algorithm to decide the intruder deduction problem for $\mathcal{I}_{\mathsf{sign}}$.

Hint: Show that $\mathcal{I}_{\mathsf{sign}}$ *is a "local" inference system, for the following notion of subterm: the set* $\mathsf{st}_{\mathsf{ext}}(t)$ *of* extended subterms *of* t *is the smallest set such that*

- $\mathsf{st}(t) \subseteq \mathsf{st}_{\mathsf{ext}}(t)$, *and*
- *if* $\mathsf{sign}(\mathsf{blind}(t_1, t_2), t_3) \in \mathsf{st}_{\mathsf{ext}}(t)$ *then* $\mathsf{sign}(t_1, t_3) \in \mathsf{st}_{\mathsf{ext}}(t)$.

Exercise 3 (***). Exhibit an inference deduction system $\mathcal{I}$ such that the intruder deduction problem associated to $\mathcal{I}$ is undecidable.

Exercise 4 (**). We propose an alternative procedure to decide the intruder deduction problem associated to $\mathcal{I}_{\mathsf{DY}}$ (defined in Section 3.2.2).

On input S and t, do:

- Decompose the terms of S as much as possible, that is, compute a fixed point S^* of S applying only decomposition inference rules.
- Check whether t can be obtained from S^* applying only composition inference rules.

1. Show that this procedure always terminate.
2. Why this procedure is not complete? That is, exhibit S and t such that the procedure fails to detect that $S \vdash_{\mathcal{I}_{\mathsf{DY}}} t$.
3. Provide additional assumptions on the inputs such that the procedure becomes complete.

Exercise 5 (**). We add to the inference system $\mathcal{I}_{\mathsf{DY}}$ a rule corresponding to block encryption modes such as ECB (Electronic codebook) or CBC (Cipher-block chaining).

$$\frac{\mathsf{aenc}(\mathsf{pair}(x, y), z)}{\mathsf{aenc}(x, z)}$$

It reflects the fact that an attacker may compute the prefix of an encrypted message (provided the length of the prefix is a multiple of the length of a block).

Show that the intruder deduction system remains decidable.

Hint: you may introduce a different notion of subterm, such that this inference system can be shown to be "local" w.r.t. this notion of subterm.

4

Equational theory and static equivalence

Inference systems model what an attacker can compute. This is however still not sufficient in cases where the attackers gains information not by learning new values but by observing the difference between two behaviors. Consider for example the case of an election where voters vote 0 or 1. An important security property is that votes remain confidential. However, the question is not whether the values 0 or 1 remain confidential since they are both public. The votes remains confidential if an attacker cannot distinguish whether some agent A is voting 0 or 1.

We first start this chapter by enriching term algebras with equational theories, to model more primitives. We then provide a formal notion of indistinguishability, called *static equivalence*.

4.1 Equational theories

A pure free algebra does not always suffice to accurately represent cryptographic primitives. Consider for example the bit-wise exclusive or operator $\oplus$. This operation has a cancellation property: $m \oplus m = 0$. This is of course true even if this operation is nested in other primitives. For example $\{m \oplus m\}_k = \{0\}_k$. The exclusive or is also commutative

and associative. These properties cannot be accurately reflected by an inference system. Instead, it is convenient to quotient the term algebra with an equational theory.

4.1.1 Definitions

Let $\mathcal{F}$ be a signature. An *equational theory* E is simply a set of *equations* $u = v$, where u and v are terms in $T(\mathcal{F}, \mathcal{X}, \mathcal{N})$.

The equivalence relation $=_E$ is defined by the equalities of E closed by reflexivity, transitivity, substitutions and context. Formally, $=_E$ is the smallest equivalence relation such that:

- $u\theta =_E v\theta$ for any $u = v \in E$ and any substitution θ,
- $u_1 =_E v_1, \ldots, u_k =_E v_k$ implies $f(u_1, \ldots, u_k) =_E f(v_1, \ldots, v_k)$.

4.1.2 Examples

Exclusive Or

The standard equational theory $E_\oplus$ for representing the exclusive or is defined as follows

$$\begin{array}{rclcrcl} x \oplus (y \oplus z) & = & (x \oplus y) \oplus z & \qquad & x \oplus y & = & y \oplus x \\ x \oplus x & = & 0 & & x \oplus 0 & = & x \end{array}$$

where $\oplus$ is a function symbol of arity 2. The two equations of the first line model resp. associativity and commutativity while the last ones model the cancellation property and the identity element of the exclusive or.

Example 4.1. Let $u = k_1 \oplus k_2$, $v = k_2 \oplus k_3$, and $w = k_1 \oplus k_3$. Then $w =_{E_\oplus} u \oplus v$.

Modular Exponentiation

Another standard primitive that cannot be modeled by an inference system is modular exponentiation. Modular exponentiation is used in many encryption schemes (such as RSA or El Gamal) and is at the heart of the Diffie-Hellman key exchange protocol [Diffie and Helman, 1976]

for example. A minimal equational theory for modular exponentiation is E_{exp} induced by the equation

$$\mathsf{exp}(\mathsf{exp}(x, y), z) = \mathsf{exp}(\mathsf{exp}(x, z), y)$$

where exp is a function symbol of arity 2.

Of course, this is not the only equation that holds for modular exponentiation but it suffices to reflect the property needed to execute the Diffie-Hellman protocol:

$$\begin{aligned} A &\to B : \ \mathsf{exp}(g, n_a) \\ B &\to A : \ \mathsf{exp}(g, n_b) \end{aligned}$$

At the end of an honest execution, the two agents A and B share the same key $\mathsf{exp}(\mathsf{exp}(g, n_a), n_b) =_{E_{\mathsf{exp}}} \mathsf{exp}(\mathsf{exp}(g, n_b), n_a)$.

Encryption

Encryption and concatenation can also be modeled by an equational theory, with the introduction of explicit *destructor* operators. Let

$$\mathcal{F}_{\mathsf{dec}} = \{\mathsf{sdec}, \mathsf{adec}, \mathsf{fst}, \mathsf{snd}\}$$

corresponding to, respectively, symmetric decryption, asymmetric decryption, first, and second projections. Let $\mathcal{F}_0$ be an arbitrary (finite) set of constant symbols. The properties of concatenation and standard symmetric and asymmetric encryption are modeled by the following set of equations E_{enc}, over the term algebra $T(\mathcal{F}_{\mathsf{std}} \cup \mathcal{F}_{\mathsf{dec}} \cup \mathcal{F}_0, \mathcal{X})$:

$$\begin{aligned} \mathsf{sdec}(\mathsf{senc}(x, y), y) &= x & \qquad \mathsf{adec}(\mathsf{aenc}(x, \mathsf{pk}(y)), y) &= x \\ \mathsf{fst}(\mathsf{pair}(x, y)) &= x & \qquad \mathsf{snd}(\mathsf{pair}(x, y)) &= y \end{aligned}$$

4.1.3 Deduction

It is again possible to model what an attacker can deduce from a given set of messages. Actually, when cryptographic primitives are modeled by an equational theory, the notion of deduction can be defined in a uniform way: the intruder may deduce any term that he can obtain

by applying functions. Formally, given an equational theory E, the associated deduction system $\vdash_E$ is defined as follows:

$$\frac{t_1 \quad \cdots \quad t_n}{f(t_1, \ldots, t_n)} \qquad \frac{t}{t'} \text{ if } t =_E t'$$

Example 4.2. Consider the equational theory $E_\oplus \cup E_{\mathsf{enc}}$ of XOR and encryption, over terms in $T(\mathcal{F}_{\mathsf{std}} \cup \mathcal{F}_{\mathsf{dec}} \cup \{\oplus\}, \mathcal{X})$. Let

$$S = \{\mathsf{senc}(a, a \oplus c), a \oplus b, b \oplus c\}.$$

Then $S \vdash_{E_\oplus \cup E_{\mathsf{enc}}} a$. Indeed

$$\dfrac{\dfrac{\mathsf{senc}(a, a \oplus c) \qquad \dfrac{a \oplus b \quad b \oplus c}{a \oplus c}}{a}}{}$$

Alternatively, deduction can be characterized as a variant of a unification problem.

Proposition 4.1. Let E be an equational theory. Let S be a set of terms and t a term. Then $S \vdash_E t$ if and only if there exist a context C such that $\mathsf{n}(C) = \emptyset$ and terms $t_1, \ldots, t_n \in S$ such that $t =_E C[t_1, \ldots, t_n]$.

A *context* is a term with holes (more formally, with variables $x_1, \ldots, x_n$) and $C[t_1, \ldots, t_n]$ denotes C where the holes (more formally the variables x_i) are replaced by the t_i.

Proof. The implication $S \vdash_E t \Rightarrow \exists C, t_1, \ldots, t_n \in S$ s.t. $\mathsf{n}(C) = \emptyset$ and $t =_E C[t_1, \ldots, t_n]$ is proved by a simple induction of the size of the proof tree of $S \vdash_E t$.

The converse implication is proved again by induction, on the size of the context C. □

This property was first noticed by Abadi and Cortier [2004] and the existence of a context C such that $t =_E C[t_1, \ldots, t_n]$ has been later called the *cap unification problem* [Anantharaman et al., 2007].

4.1.4 The case of explicit decryption

Back to the case of encryption, there are two ways of defining deduction: either through the inference system $\mathcal{I}_{\mathsf{DY}}$ introduced in §3.2.1 or

through the deduction system induced by the equational theory E_{enc}. Both definitions actually coincide.

Proposition 4.2. Let S be a set of terms and t be a term in the term algebra $T(\mathcal{F}_{\mathsf{std}}, \mathcal{X})$. Then

$$S \vdash_{\mathcal{I}_{\mathsf{DY}}} t \quad \text{if and only if} \quad S \vdash_{E_{\mathsf{enc}}} t.$$

The proof is left as an exercise.

4.2 Static equivalence

Deduction does not always suffice to reflect the attacker's abilities. For example, the attacker can observe in which order messages are sent. Messages are therefore organized into *frames*.

Definition 4.1 (frame)**.** A frame is an expression $\varphi = \nu\tilde{n}\theta = \nu\tilde{n}\{M_1/x_1, \ldots, M_n/x_n\}$ where θ is a substitution and $\tilde{n}$ is a set of names that are restricted in φ. The terms $M_1, \ldots, M_n$ represent the attacker knowledge while the names in $\tilde{n}$ are initially unknown to the attacker. If k is a name $\nu k\varphi$ denotes $\nu(\tilde{n} \cup \{k\})\theta$. We define the domain of φ, $\mathsf{Dom}(\varphi)$, to be $\mathsf{Dom}(\theta)$.

Example 4.3. $\nu k\{1/x_1, 0/x_2, \mathsf{senc}(0,k)/x_3\}$ is a frame. It models the fact that the attacker has seen two constants 1 and 0 in addition to the encryption $\mathsf{senc}(0,k)$ where the key k is initially unknown to the attacker.

4.2.1 Deduction and recipe

The notion of deduction can easily be extended to frames.

Definition 4.2 (deduction)**.** A term t is deducible from a frame $\varphi = \nu\tilde{n}\theta$ if it can be deduced using φ and any name that does not occur in $\tilde{n}$. More formally, given an equational theory E and a frame $\varphi = \nu\tilde{n}\theta$, we write $\varphi \vdash_E t$ if

$$\mathsf{Dom}(\varphi) \cup (\mathcal{N} \backslash \tilde{n}) \vdash_E t.$$

Consider for example the frame $\varphi_1 = \nu n, k\theta_1$ where

$$\theta_1 = \{\mathsf{senc}(\mathsf{pair}(n,n),k)/x, k/y\}$$

and consider the equational theory E_{enc} defined in §4.1.2. Then $\varphi_1 \vdash_{E_{\mathsf{enc}}} n$. More precisely, n can be obtained from φ_1 by first decrypting the message stored in x by the key in y and then projecting on the first component. Let $M = \mathsf{fst}(\mathsf{dec}(x,y))$. We have that $M\theta_1 =_{E_{\mathsf{enc}}} n$. Such a term M is called a *recipe* of n w.r.t. φ_1.

Definition 4.3 (recipe)**.** Let $\varphi = \nu\tilde{n}\theta$ be a frame and t a term such that $\varphi \vdash_E t$. A term R is said free w.r.t. φ if $\mathsf{n}(R) \cap \tilde{n} = \emptyset$. A term R is a *recipe* of t in φ if R is free w.r.t. φ and if $R\theta =_E t$.

A term is deducible if and only if there exists a recipe of it.

Proposition 4.3. Let $\varphi = \nu\tilde{n}\theta$ be a frame and t a term. $\varphi \vdash_E t$ if and only if there exists a recipe R of t in φ.

The proof is very similar to the proof of Proposition 4.1.

4.2.2 Definition of static equivalence

Consider the two frames $\varphi_1 = \{0/x, 1/y\}$ and $\varphi_2 = \{1/x, 0/y\}$ where 0 and 1 are constants. Clearly, the same terms can be deduced from φ_1 and φ_2. However, the two frames model two different situations. In φ_1, the attacker observed 0 and 1, while in φ_2, the attacker observed 1 then 0. To reflect the ability of the attacker to *compare* messages, Abadi and Fournet [2001] introduced the notion of *static equivalence*. Before defining static equivalence we require the following auxiliary definition.

Definition 4.4. Given frames φ_1, φ_2 we write $\varphi_1 =_\alpha \varphi_2$ if φ_1 is equal to φ_2 up to α-conversion of restricted names.

We say that the equation $M =_E N$ holds in a frame φ, written $(M =_E N)\varphi$ if and only if there exist $\tilde{n}$ and θ such that $\varphi =_\alpha \nu\tilde{n}\theta$, M, N are free w.r.t. $\nu\tilde{n}\theta$ and $M\theta =_E N\theta$.

Definition 4.5 (static equivalence)**.** Two frames φ_1 and φ_2 are statically equivalent w.r.t. an equational theory E, denoted $\varphi_1 \sim_E \varphi_2$, if $\mathsf{Dom}(\varphi_1) = \mathsf{Dom}(\varphi_2)$ and for any two terms M, N we have that

$$(M =_E N)\varphi_1 \text{ if and only if } (M =_E N)\varphi_2.$$

We may write $\varphi_1 \sim \varphi_2$ instead of $\varphi_1 \sim_E \varphi_2$ when E is clear from the context.

Example 4.4. Let E_{enc} be the equational theory of encryption as defined in §4.1.2. Let $\varphi_1 = \{0/x, 1/y\}$ and $\varphi_2 = \{1/x, 0/y\}$. Then $\varphi_1 \not\sim \varphi_2$. Indeed $(x = 0)\varphi_1$ while $(x \neq 0)\varphi_2$.

Example 4.5. Let $\varphi_1 = \nu k\{\mathsf{aenc}(0, \mathsf{pk}(k))/x, \mathsf{pk}(k)/y\}$ and $\varphi_2 = \nu k\{\mathsf{aenc}(1, \mathsf{pk}(k))/x, \mathsf{pk}(k)/y\}$ corresponding respectively to the asymmetric encryption of 0 and 1. Then $\varphi_1 \not\sim \varphi_2$. Indeed $(\mathsf{aenc}(0, y) = x)\varphi_1$ while $(aenc(0, y) \neq x)\varphi_2$. An attacker may encrypt 0 itself and checks for equality.

This is not the case anymore if encryption is randomized. Let $\varphi'_1 = \nu k, n\{\mathsf{aenc}(\mathsf{pair}(0, n), \mathsf{pk}(k))/x, \mathsf{pk}(k)/y\}$ and $\varphi'_2 = \nu k, n\{\mathsf{aenc}(\mathsf{pair}(1, n), \mathsf{pk}(k))/x, \mathsf{pk}(k)/y\}$. Then $\varphi'_1 \sim \varphi'_2$.

Static equivalence is closed under restriction and composition.

Proposition 4.4. Let $\varphi = \nu\tilde{n}\sigma$, $\varphi_1 = \nu\tilde{n}_1\sigma_1$, and $\varphi_2 = \nu\tilde{n}_2\sigma_2$ be three frames such that $\varphi_1 \sim \varphi_2$, $\mathsf{Dom}(\sigma) \cap \mathsf{Dom}(\sigma_i) = \emptyset$ and $\tilde{n} \cap \tilde{n}_i = \emptyset$ $(1 \leq i \leq 2)$. Then

1. $\nu s\ \varphi_1 \sim \nu s\ \varphi_2$, and
2. $\psi_1 = \nu(\tilde{n} \cup \tilde{n}_1)(\sigma \cup \sigma_1) \sim \nu(\tilde{n} \cup \tilde{n}_1)\sigma \cup \sigma_2 = \psi_2$

Proof. Property 1 follows from the fact that $\mathsf{n}(\nu s\ \varphi_i) \subseteq \mathsf{n}(\varphi_i)$. Property 2 is also simple to prove. Assume $(M = N)\psi_1$. This can be rewritten as $(M\sigma = N\sigma)\varphi_1$ assuming $\mathsf{n}(M, N) \cap \tilde{n} = \emptyset$, which can be enforced through α-renaming. Since $\varphi_1 \sim \varphi_2$, this implies $(M\sigma = N\sigma)\varphi_2$, therefore $(M = N)\psi_2$. The case where $(M = N)\psi_2$ is symmetric. □

4.2.3 Decision procedure for encryption

In this section, we consider the equational theory of symmetric encryption E_{senc} restricting the theory E_{enc} (defined in §4.1.2) to pair and symmetric encryption. Our goal is to present a decision procedure for static equivalence, for the theory E_{senc}. The procedure is a special case of the procedure presented by Baudet [2005], for arbitrary subterm convergent equational theories.

For the sake of clarity of exposition, we consider only frames where all names are restricted (public names should be explicitly added in the frame). That is, we only consider frames of the form $\nu\tilde{n}\theta$ where $\mathsf{n}(\theta) \subseteq \tilde{n}$. Therefore, by abuse of notations, we write $\theta_1 \sim_{E_{\mathsf{senc}}} \theta_2$ whenever $\nu\mathsf{n}(\theta_1)\theta_1 \sim_{E_{\mathsf{senc}}} \nu\mathsf{n}(\theta_2)\theta_2$. We first need to introduce some vocabulary.

Definition 4.6 (extended frame)**.** An *extended frame* is an expression $\{M_1 \triangleright u_1, \ldots, M_n \triangleright u_n\}$ where u_i and M_i are terms. The extended frame associated to a frame $\nu\tilde{n}\{u_1/x_1, \ldots, u_n/x_n\}$ (where $\mathsf{n}(u_i) \subseteq \tilde{n}$) is simply defined as $\{x_1 \triangleright u_1, \ldots, x_n \triangleright u_n\}$.

Initialization Given a substitution θ, the decision procedure starts by eliminating duplicated terms, replacing them by equations. Suppose the extended frame corresponding to θ is

$$\{x_1^1 \triangleright t_1, \ldots, x_1^{k_1} \triangleright t_1, \ldots, x_n^1 \triangleright t_n, \ldots, x_n^{k_n} \triangleright t_n\}$$

where the t_i are pairwise distinct. Then we define

$$\mathsf{Init}(\theta) = (\{x_1^1 \triangleright t_1, x_2^1 \triangleright t_2, \ldots, x_n^1 \triangleright t_n\}, \{x_i^1 \diamond x_i^j \mid 1 \leq i \leq n, 1 \leq j \leq k_i\})$$

Saturation Let φ_0 be an extended frame and E_0 a set of equations $M \diamond N$ where M and N are terms. We define the saturation procedure for φ_0 and E_0 as follows.
$\mathsf{Sat}(\varphi_0, E_0) =$

1. Let $\varphi := \varphi_0$ and $E := E_0$.
2. Repeat until reaching a fixed point.

   For any $M_1 \triangleright t_1, M_2 \triangleright t_2 \in \varphi$, $f \in \{\mathsf{senc}, \mathsf{dec}, \mathsf{pair}\}$, $g \in \{\mathsf{fst}, \mathsf{snd}\}$:

   - If $f(t_1, t_2) =_{E_{\mathsf{senc}}} t$ for some term t subterm of φ then
     - if $\exists M.\ M \triangleright t \in \varphi$ then $E := E \cup \{f(M_1, M_2) \diamond M\}$;
     - else $\varphi := \varphi \cup \{f(M_1, M_2) \triangleright t\}$.
   - If $g(t_1) =_{E_{\mathsf{senc}}} t$ for some term t subterm of φ then
     - if $\exists M.\ M \triangleright t \in \varphi$ then $E := E \cup \{g(M_1) \diamond M\}$;
     - else $\varphi := \varphi \cup \{g(M_1) \triangleright t\}$.

3. Return E.

This procedure terminates and reaches a fixed point (left as exercise).

Theorem 4.1. Let θ_1 and θ_2 be two substitutions. then $\theta_1 \sim_{E_{\mathsf{senc}}} \theta_2$ if and only if

- for any $(M \diamond N) \in \mathsf{Sat}(\mathsf{Init}(\theta_1))$ it holds that $M\theta_2 =_{E_{\mathsf{senc}}} N\theta_2$.
- for any $(M \diamond N) \in \mathsf{Sat}(\mathsf{Init}(\theta_2))$ it holds that $M\theta_1 =_{E_{\mathsf{senc}}} N\theta_1$.

This theorem is a particular case of the decision procedure developed by Baudet [2005]. Its proof will not be given here.

Example 4.6. Consider $\theta = \{\mathsf{senc}(\mathsf{pair}(n,n),k)/x, k/y\}$. Then

$$\mathsf{Init}(\theta) = (\{x \triangleright \mathsf{senc}(\mathsf{pair}(n,n),k), y \triangleright k\}, \emptyset)$$

$$\text{and } \mathsf{Sat}(\mathsf{Init}(\theta)) = (\varphi, E)$$

with

$$\varphi = x \triangleright \mathsf{senc}(\mathsf{pair}(n,n),k), y \triangleright k, \mathsf{dec}(x,y) \triangleright \mathsf{pair}(n,n), \mathsf{fst}(\mathsf{dec}(x,y)) \triangleright n$$

$$\text{and } E = \{\mathsf{snd}(\mathsf{dec}(x,y)) \diamond \mathsf{fst}(\mathsf{dec}(x,y))\}$$

4.2.4 More decision procedures

A much more general decision procedure has been developed by Baudet [2005] for any convergent theory. It is guaranteed to terminate for the class of convergent subterm theories[1] and it also works in more general cases that encompass for example blind signatures [Baudet et al., 2013]. The corresponding implementation is the tool YAPA. Another available tool for static equivalence is KISS by Ciobâcă et al. [2012]. This tool also handles a large class of equational theories that includes for example trapdoor commitments.

[1] A *convergent* theory is an equational theory induced by a convergent rewriting system. The theory is *subterm convergent* if there is a corresponding (convergent) rewriting system such that any rewrite rule $\ell \to r$ is such that r is a subterm of ℓ or a constant.

Decidability of static equivalence has also been obtained for the class of *monoidal theories* [Cortier and Delaune, 2012] that captures e.g. Exclusive Or and pure Associativity and Commutativity. It also encompasses homomorphic equations of the form $h(x+y) = h(x)+h(y)$ where $+$ is an associative and commutative symbol. Decidability results can be combined [Cortier and Delaune, 2012] : if static equivalence is decidable for two disjoint theories E_1 and E_2 then it is also decidable for $E_1 \cup E_2$ (provided deduction is also decidable for E_1 and E_2).

Some more decidability results have been obtained for specific theories such as the theory of trapdoor commitment and that of reencryption [Berrima et al., 2011] as well as theories for Diffie-Hellman exponentiation [Kremer and Mazaré, 2007] and bilinear pairings [Kremer et al., 2012].

4.3 Exercises

Exercise 6 (**). Consider the equational theory $E_{\oplus}$ of the Exclusive Or, defined in §4.1.2. Show that the following problem is decidable.

Input a finite set of terms S and a term t over $T(\{\oplus\}, \mathcal{N})$

Output whether $S \vdash_{E_{\oplus}} t$.

Discuss the complexity of your algorithm (Note: there exists a polynomial time algorithm).

Exercise 7 (**). Consider the equational theory E_{AC} of associativity and commutativity, defined by the two following equations.

$$\{(x + y) + z = x + (y + z), \qquad x + y = y + x\}$$

Show that the following problem is decidable.

Input a finite set of terms S and a term t over $T(\{+\}, \mathcal{N})$

Output whether $S \vdash_{E_{+}} t$.

Discuss the complexity of your algorithm (Note: this problem is actually NP-complete).

Exercise 8 (***). Let S be a set of terms and t be a term in the term algebra $T(\mathcal{F}_{\mathsf{std}}, \mathcal{X})$. Show that

$$S \vdash_{\mathcal{I}_{\mathsf{DY}}} t \quad \text{if and only if} \quad S \vdash_{E_{\mathsf{enc}}} t.$$

Exercise 9 (*). Let $\varphi_1 = \nu\tilde{n}_1\theta_1$ and $\nu\varphi_2 = \tilde{n}_2\theta_2$. We define $\varphi_1 \overset{\circ}{\sim}_E \varphi_2$ if and only if

- $\mathsf{Dom}(\varphi_1) = \mathsf{Dom}(\varphi_2)$, and
- for all M, N that are free in φ_1 and φ_2 we have that

$$(M\theta_1) =_E (N\theta_1) \text{ iff } (M\theta_2) =_E (N\theta_2)$$

Show that static equivalence ($\sim_E$) and $\overset{\circ}{\sim}_E$ do *not* coincide.

Exercise 10 (**). Show that the saturation procedure described in §4.2.3 terminates.

Exercise 11 (*). Let $\theta_1 = \{\mathsf{senc}(\mathsf{pair}(n,n),k)/x, k/y\}$, $\theta_2 = \{\mathsf{senc}(\mathsf{pair}(\mathsf{senc}(n,k'),\mathsf{senc}(n,k')),k)/x, k/y\}$, and $\theta_2 = \{\mathsf{senc}(\mathsf{pair}(n,n'),k)/x, k/y\}$,

1. Show that $\theta_1 \not\sim_{E_{\mathsf{senc}}} \theta_3$.
2. Show that $\theta_1 \sim_{E_{\mathsf{senc}}} \theta_2$.

   Hint: you may use the decision procedure of §4.2.3.

Exercise 12 (***). Consider the theory E_{enc} defined in §4.1.2. Let h be a unary symbol (that has no equation). Show that for any ground term t,

$$\nu n.\ \{^{h(n)}/_{x}\} \sim_{E_{\mathsf{enc}}} \nu n.\ \{^{h(\mathsf{pair}(t,n))}/_{x}\}$$

5

A cryptographic process calculus

The previous chapter describes how messages exchanged in cryptographic protocols can be represented as *terms*. In this chapter, we discuss how the protocols themselves can be modelled. While the kind of "Alice - Bob" notation used in §2 are convenient for explaining protocols, these notations generally only describe a honest protocol execution, and contain ambiguities. Fundamentally, security protocols are concurrent programs and formalisms for representing such programs do exist. In particular process algebras and calculi have been developed for this purpose. Some "general purpose process algebras", e.g. CSP [Ryan et al., 2000], have indeed been used to reason about security protocols. There also exist dedicated calculi which integrate support for sending messages that use cryptographic primitives. Examples of such dedicated process calculi are CryptoSPA [Focardi and Martinelli, 1999], which extend the CCS process algebra, the spi calculus [Abadi and Gordon, 1999], and the applied pi calculus [Abadi and Fournet, 2001], that both extend the pi calculus. We present here in more details the applied pi calculus. In contrast to the pure pi calculus it is not restricted to communicating names, but processes may output terms that represent messages. One may also note that some people have con-

$P, Q, R :=$ plain processes
- $\mathsf{0}$
- $P \parallel Q$
- $!P$
- $\nu n.P$
- $\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q$
- $\mathsf{in}(u, x).P$
- $\mathsf{out}(u, t).P$

Figure 5.1: Syntax: plain processes

sidered the problem of compiling "Alice - Bob" kind notations to more formal models, e.g., [Jacquemard et al., 2000, Millen and Denker, 2002, Chevalier and Rusinowitch, 2010], but we will not discuss them here.

5.1 Syntax and informal semantics

We assume a set of names $\mathcal{N}$, a set of variables $\mathcal{X}$, and a signature $\mathcal{F}$ which define the set of terms $T(\mathcal{F}, \mathcal{X}, \mathcal{N})$ equipped with an equational theory E (see §4). The equational theory is left implicit throughout this chapter. Moreover, we rely on a simple sort system that distinguishes channels and basic messages. We distinguish the set $\mathcal{N}_{ch} \subset \mathcal{N}$ of channel names and partition $\mathcal{X} = \mathcal{X}_b \uplus \mathcal{X}_{ch}$ in the set of variables of base sort and variables of channel sort. We suppose that function symbols cannot be applied to variables or names of channel sort, and cannot return terms of that sort. Hence, channels are always atomic: the only terms of channel sort are variables and names of that sort.

The applied pi calculus has two kind of processes: *plain* and *extended* processes. Plain processes are generated by the grammar given in Figure 5.1, where $t, t_1, t_2, \ldots$ range over terms, n over names, x over variables and u is a meta-variable that stands for either a name or a variable of channel sort. The $\mathsf{0}$ process is the process that does nothing. Parallel composition $P \parallel Q$ models that processes P and Q are executed in parallel. The replication of P, denoted $!P$, allows an unbounded number of copies of P to be spawned. New names can be

$A, B, C :=$ extended processes
P
$A \parallel B$
$\nu n.A$
$\nu x.A$
$\{^t/_x\}$

Figure 5.2: Syntax: extended processes

created using the *new* operator νn, which acts as a binder and generates a *restricted* name. The conditional $\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q$ behaves as P whenever $t_1 =_E t_2$ and as Q otherwise. Finally, $\mathsf{in}(u, x).P$ expects an input on channel u that is bound to variable x in P and $\mathsf{out}(u, M).P$ outputs term M on channel u and then behaves as P.

Extended processes are generated by the grammar given in Figure 5.2. They extend plain processes by *active substitutions*, and allow restrictions on both names and variables. An active substitution $\{^t/_x\}$ allows processes to address a term by a variable. The scope of this access may be restricted using the ν operator on variables. This also allows to define local variables as follows: the construct $\mathsf{let}\ x = t\ \mathsf{in}\ P$ is defined as $\nu x.(P \parallel \{^t/_x\})$. When the variable x is not restricted, it means that the environment, which represents the attacker, may use x to access the term t. As exemplified in the description of the labelled semantics of the applied pi calculus, these active substitutions are typically used to record the terms output by the processes and represent the attacker knowledge. Given several active substitutions in parallel $\{^{t_1}/_{x_1}\} \parallel \ldots \parallel \{^{t_n}/_{x_n}\}$ we often regroup them in a single substitution $\{^{t_1}/_{x_1}, \ldots, ^{t_n}/_{x_n}\}$. We suppose that these substitutions are cycle free, that there is at most one substitution defined for each variable, and exactly one when this variable is restricted, and substitutions only contain terms of base sort. Given an extended process A we denote by $\phi(A)$ the process obtained by replacing any plain process with 0. $\phi(A)$ is called the frame of the process A. We also note that extended processes must not appear under a replication, an input, an output, or a conditional.

For readability we often omit trailing 0 processes and else 0 branches, e.g. we write

$$\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ \mathsf{out}(c, t)$$

instead of

$$\mathsf{if}\ t_1 = t_2\ \mathsf{then}\ \mathsf{out}(c, t).0\ \mathsf{else}\ 0$$

As usual we define free and bound names for processes, denoted $\mathsf{n}(A)$ and $\mathsf{bn}(A)$. Similarly we define free and bound variables, denoted $\mathsf{fv}(A)$ and $\mathsf{bv}(A)$. The set of bound variables contains all variables that have been bound by an input and the ones that are restricted by the ν operator. For an active substitution $\{^t/_x\}$ the set of free variables contains x in addition to the variables occurring in t.

Finally, we define the notion of *context*. A context is a process with a "hole", often denoted _. Given a context C, we write $C[A]$ for the process obtained by replacing _ with the process A. An evaluation context is a context whose hole is neither under replication, nor input, output or conditional.

5.2 Modelling protocols as processes

Before defining the formal semantics of the applied pi calculus we illustrate how the calculus can be used for modelling security protocols. As an example we consider the Needham-Schroeder public key protocol, introduced in §2. The protocol can be informally described as follows.

1. $A \to B$: $\mathsf{aenc}(\langle a, n_a\rangle, \mathsf{pk}_b)$
2. $B \to A$: $\mathsf{aenc}(\langle n_a, n_b\rangle, \mathsf{pk}_a)$
3. $A \to B$: $\mathsf{aenc}(n_b, \mathsf{pk}_b)$

We assume the previously defined equational theory E_{enc} on signature $\mathcal{F}_{\mathsf{dec}} \cup \mathcal{F}_{\mathsf{std}}$ and model each of the roles A and B by a process. It is important to distinguish between the *role* of the initiator A, modelled by a process, and the agent (a in the above informal description) who is executing the role. To make this distinction explicit we parametrize the processes representing the initiator and responder with the keys of the agents who execute the role.

$$\begin{array}{rcl} P_A(sk_i, pk_r) & \hat{=} & \nu n_a.\ \mathsf{out}(c, \mathsf{aenc}(\langle \mathsf{pk}(sk_i), n_a \rangle, \mathsf{pk}_r)). \\ & & \mathsf{in}(c, x). \\ & & \mathsf{if}\ \mathsf{fst}(\mathsf{adec}(x, sk_i)) = n_a\ \mathsf{then} \\ & & \mathsf{let}\ x_{nb} = \mathsf{snd}(\mathsf{adec}(x, sk_i))\ \mathsf{in} \\ & & \mathsf{out}(c, \mathsf{aenc}(x_{nb}, \mathsf{pk}_r)) \end{array}$$

The process first generates a fresh nonce n_a and then outputs the first message on channel c. Note that for simplicity we assume that the agent's identity is his public key. Next, the initiator is waiting for a message which is going to be bound to the variable x. Using a conditional the initiator checks that the message contains the previously sent nonce n_a. For readability we then create a local variable x_{nb} and store in this variable what is expected to be the nonce generated by the responder.

We can model similarly the responder process P_B.

$$\begin{array}{rcl} P_B(sk_r) & \hat{=} & \mathsf{in}(c, y). \\ & & \mathsf{let}\ pk_i = \mathsf{fst}(\mathsf{adec}(y, sk_r))\ \mathsf{in} \\ & & \mathsf{let}\ y_{na} = \mathsf{snd}(\mathsf{adec}(y, sk_r))\ \mathsf{in} \\ & & \nu n_b.\mathsf{out}(c, \mathsf{aenc}(\langle y_{na}, n_b \rangle, pk_i)) \\ & & \mathsf{in}(c, z). \\ & & \mathsf{if}\ \mathsf{adec}(z, sk_r)) = n_b\ \mathsf{then}\ Q \end{array}$$

One may note that P_B only takes a single argument, the responder's private key. The initiator's public key is received during the execution. When the final test succeeds we suppose that the responder continues to execute some task modelled by the process Q.

We can now put the processes together into a process that models the Needham Schroeder public key protocol as a whole.

$$\begin{array}{rl} P^1_{\mathsf{nspk}} \hat{=} \nu sk_a, sk_b.(P_A(sk_a, \mathsf{pk}(sk_b)) \parallel P_B(sk_b) \parallel & \\ \qquad \mathsf{out}(c, \mathsf{pk}(sk_a)) \parallel \mathsf{out}(c, \mathsf{pk}(sk_b))) & \end{array}$$

This first version models that a (or more precisely the agent identified by $\mathsf{pk}(sk_a)$) is executing an instance of the role P_A with b (identified by $\mathsf{pk}(sk_b)$). We also output the public keys of a and b to make these available to the adversary.

However, one may notice that the above modeling would miss Lowe's man in the middle attack since this setting does not involve any dishonest agent c. To capture this attack one could explicitly include that a is willing to start a session with the intruder. We suppose that the intruder possesses a secret key sk_c which formally is just a free name.

$$P^2_{\mathsf{nspk}} \hat{=} \nu sk_a, sk_b.(P_A(sk_a, \mathsf{pk}(sk_b)) \parallel P_A(sk_a, \mathsf{pk}(sk_c)) \parallel P_B(sk_b) \parallel \mathsf{out}(c, \mathsf{pk}(sk_a)) \parallel \mathsf{out}(c, \mathsf{pk}(sk_b)))$$

This second version explicitly includes a session started by a with the intruder and indeed captures Lowe's man in the middle attack. However, this situation is not satisfactory, as one does not know a priori with whom agents should start a session. One trick is to leave this choice to the attacker: we add an input that is used to define the public key given to the initiator role.

$$P^3_{\mathsf{nspk}} \hat{=} \nu sk_a, sk_b.(\mathsf{in}(c, x_{pk}).P_A(sk_a, x_{pk}) \parallel P_B(sk_b) \parallel \mathsf{out}(c, \mathsf{pk}(sk_a)) \parallel \mathsf{out}(c, \mathsf{pk}(sk_b)))$$

Now the attacker can just input the public key that suits him best to create an attack. Note that he may input one of the two regular public keys $\mathsf{pk}(sk_a)$ and $\mathsf{pk}(sk_b)$, or any other term, including in particular his own key $\mathsf{pk}(sk_c)$. Note that the attacker may also trigger the agent a to execute a protocol "with himself", i.e., with the public key $\mathsf{pk}(sk_a)$. There exist indeed attacks, sometimes called *reflection attacks*, that rely on this behavior.

This version has still a shortcoming. We only consider one session for each role. Many attacks do however require several parallel sessions of the same role. Millen [1999] has even shown that there is a priori no upper bound on the number of parallel sessions that would avoid all attacks. We therefore add replication.

$$P^4_{\mathsf{nspk}} \hat{=} \nu sk_a, sk_b.(!\mathsf{in}(c, x_{pk}).P_A(sk_a, x_{pk}) \parallel !P_B(sk_b) \parallel \mathsf{out}(c, \mathsf{pk}(sk_a)) \parallel \mathsf{out}(c, \mathsf{pk}(sk_b)))$$

This new version allows a and b to execute an arbitrary number of sessions. Note that a may execute several sessions with the same as well

as different responders. However, this modeling still misses that both initiator and responder may be executed by the same agent. We therefore include explicitly that a and b may execute both roles. Moreover, the above process only allows two honest agents while an attack may a priori require the presence of more honest agents. Therefore we add an additional replication that allows the creation of an arbitrary number of honest private keys, each of which can be used in an arbitrary number of sessions.

$$\begin{aligned} P^5_{\mathsf{nspk}} \triangleq\ !\nu sk_a, sk_b.(!\mathsf{in}(c, x_{pk}).P_A(sk_a, x_{pk}) \parallel !P_B(sk_a) \parallel \\ !\mathsf{in}(c, x_{pk}).P_A(sk_b, x_{pk}) \parallel !P_B(sk_b) \parallel \\ \mathsf{out}(c, \mathsf{pk}(sk_a)) \parallel \mathsf{out}(c, \mathsf{pk}(sk_b))) \end{aligned}$$

Observing the symmetric roles of a and b this process can be written more succinctly and elegantly as

$$P^6_{\mathsf{nspk}} \triangleq\ !\nu sk.(!\mathsf{in}(c, x_{pk}).P_A(sk, x_{pk}) \parallel !P_B(sk) \parallel \mathsf{out}(c, \mathsf{pk}(sk)))$$

This final modeling allows the adversary to spawn an arbitrary number of instances of P_A and P_B with either the same or different private keys.

5.3 Formal semantics

As the goal is to prove security properties of protocols modelled as processes, we need to define the semantics of the calculus in order to have a precise definition on how a process can be executed.

5.3.1 Operational semantics

We first define the notion of *structural equivalence*. Intuitively, structural equivalence relates identical processes that are simply written in a different way.

Formally, structural equivalence $\equiv$ is the smallest equivalence relation closed under α-conversion of bound names and variables and application of evaluation contexts and such that:

Par-0	$A \parallel 0$	$\equiv$	A	
Par-C	$A \parallel B$	$\equiv$	$B \parallel A$	
Par-A	$(A \parallel B) \parallel C$	$\equiv$	$A \parallel (B \parallel C)$	
Repl	$!P$	$\equiv$	$P \parallel !P$	
New-0	$\nu\, n.\, 0$	$\equiv$	0	
New-Par	$A \parallel \nu\, u.B$	$\equiv$	$\nu\, u.(A \parallel B)$	when $u \notin \mathsf{fv}(A) \cup \mathsf{n}(A)$
New-C	$\nu\, u.\nu\, v.A$	$\equiv$	$\nu\, v.\nu\, u.A$	
Alias	$\nu\, x.\{^{t}/_{x}\}$	$\equiv$	0	
Subst	$\{^{t}/_{x}\} \parallel A$	$\equiv$	$\{^{t}/_{x}\} \parallel A\{^{t}/_{x}\}$	
Rewrite	$\{^{t_1}/_{x}\}$	$\equiv$	$\{^{t_2}/_{x}\}$	when $t_1 =_E t_2$

While most of the above rules are standard the last three rules may require some explanation. Alias allows the creation of a new local variable. Subst allows the application of an active substitution to a process and Rewrite allows to relate two active substitutions modulo the equational theory.

Example 5.1. Let us illustrate these rules by showing that $\mathsf{out}(c, t_1) \equiv \mathsf{out}(c, t_2)$ when $t_1 =_E t_2$.

$\mathsf{out}(c, t_1)$	$\equiv$	$\mathsf{out}(c, t_1) \parallel 0$	by Par-0
	$\equiv$	$\mathsf{out}(c, t_1) \parallel \nu\, x.\{^{t_1}/_{x}\}$	by Alias
	$\equiv$	$\nu\, x.(\mathsf{out}(c, t_1) \parallel \{^{t_1}/_{x}\})$	by New-Par
	$\equiv$	$\nu\, x.(\{^{t_1}/_{x}\} \parallel \mathsf{out}(c, t_1))$	by Par-C
	$\equiv$	$\nu\, x.(\{^{t_1}/_{x}\} \parallel \mathsf{out}(c, x))$	by Subst
	$\equiv$	$\nu\, x.(\{^{t_2}/_{x}\} \parallel \mathsf{out}(c, x))$	by Rewrite
	$\equiv$	$\nu\, x.(\{^{t_2}/_{x}\} \parallel \mathsf{out}(c, t_2))$	by Subst
	$\equiv$	$\nu\, x.(\mathsf{out}(c, t_2) \parallel \{^{t_2}/_{x}\})$	by Par-C
	$\equiv$	$\mathsf{out}(c, t_2) \parallel \nu\, x.\{^{t_2}/_{x}\}$	by New-Par
	$\equiv$	$\mathsf{out}(c, t_2) \parallel 0$	by Alias
	$\equiv$	$\mathsf{out}(c, t_2)$	by Par-0

Note that we also implicitly used the fact that structural equivalence is closed under application of evaluation contexts as we applied some of the rules directly under a context.

One may also note that for any extended process A, we have that $\phi(A) \equiv \nu\tilde{n}.\sigma$ for some sequence of names $\tilde{n}$ and substitution σ. Therefore we can lift static equivalence to processes and we write $A \sim_E B$ whenever $\phi(A) \equiv \nu\tilde{n}_A.\sigma_A$, $\phi(B) \equiv \nu\tilde{n}_B.\sigma_B$, and $\tilde{n}_A.\sigma_A \sim \nu\tilde{n}_B.\sigma_B$.

We can now define how processes interact together. *Internal reduction* is the smallest relation on processes closed under structural equivalence and application of evaluation contexts such that

$$\begin{array}{ll} \textsc{Comm} & \mathsf{out}(c,t).P_1 \parallel \mathsf{in}(c,x).P_2 \rightarrow P_1 \parallel P_2\{^t/_x\} \\ \textsc{Then} & \mathsf{if}\ t = t\ \mathsf{then}\ P\ \mathsf{else}\ Q \rightarrow P \\ \textsc{Else} & \mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q \rightarrow Q \\ & \qquad \text{where } t_1, t_2 \text{ are ground and } t_1 \neq_E t_2 \end{array}$$

The first rule (Comm) models communication: whenever a process is ready to output a term t on channel c and another process, running in parallel, is ready to input on channel c, i.e., it starts with $\mathsf{in}(c,x)$ then a communication can take place and x is replaced by t. Rules Then and Else model a conditional. One may note that the Then rule requires syntactic equality of terms ($\mathsf{if}\ t = t$). However as internal reduction is closed under structural equivalence this rule is equivalent to the rule

$$\begin{array}{ll} \textsc{Then'} & \mathsf{if}\ t_1 = t_2\ \mathsf{then}\ P\ \mathsf{else}\ Q \rightarrow P \\ & \qquad \text{where } t_1 =_E t_2 \end{array}$$

using structural equivalence in a similar way as in Example 5.1. One may also note that in the Else rule, contrary to the Then rule we require t_1, t_2 to be ground. This is due to the fact that equality is closed under substitution while disequality is not, e.g. even though $x \neq y$ we have that $x\sigma = y\sigma$ for $\sigma = \{^x/_y\}$.

Example 5.2. We illustrate internal reduction by modelling the honest execution of the Needham Schroeder public key protocol. For simplicity,

we consider a naive model that only considers two honest participants:

$$
\begin{array}{lll}
 & \nu sk_a, sk_b.\ P_A(sk_a, \mathsf{pk}(sk_b)) \parallel P_B(sk_b) & \\
\rightarrow & \nu sk_a, sk_b, n_a, n_b. & \mathsf{in}(c, x). \\
 & & \mathsf{if}\ \mathsf{fst}(\mathsf{adec}(x, sk_i)) = n_a\ \mathsf{then} \\
 & & \mathsf{let}\ x_{nb} = \mathsf{snd}(\mathsf{adec}(x, sk_i))\ \mathsf{in} \\
 & & \mathsf{out}(c, \mathsf{aenc}(x_{nb}, \mathsf{pk}_r)) \\
 & \parallel & \mathsf{out}(c, \mathsf{aenc}(\langle n_a, n_b \rangle, \mathsf{pk}(sk_a))) \\
 & & \mathsf{in}(c, z). \\
 & & \mathsf{if}\ \mathsf{adec}(z, sk_b)) = n_b\ \mathsf{then}\ Q \\
\rightarrow & \nu sk_a, sk_b, n_a, n_b. & \mathsf{if}\ n_a = n_a\ \mathsf{then} \\
 & & \mathsf{out}(c, \mathsf{aenc}(n_b, \mathsf{pk}_r)) \\
 & \parallel & \mathsf{in}(c, z). \\
 & & \mathsf{if}\ \mathsf{adec}(z, sk_b)) = n_b\ \mathsf{then}\ Q \\
\rightarrow & \nu sk_a, sk_b, n_a, n_b. & \mathsf{out}(c, \mathsf{aenc}(n_b, \mathsf{pk}_r)) \\
 & \parallel & \mathsf{in}(c, z). \\
 & & \mathsf{if}\ \mathsf{adec}(z, sk_b)) = n_b\ \mathsf{then}\ Q \\
\rightarrow & \nu sk_a, sk_b, n_a, n_b. & \mathsf{if}\ n_b = n_b\ \mathsf{then}\ Q \\
\rightarrow & \nu sk_a, sk_b, n_a, n_b. & Q
\end{array}
$$

5.3.2 Observational equivalence

In order to model security properties we often rely on the notion of *observational equivalence*. Examples of security properties relying on observational equivalence are provided in §6. Intuitively, two processes are observationally equivalent if they cannot be distinguished by an attacker. Here the attacker may be an arbitrary process written in the applied pi calculus. The formal definition of observational equivalence uses the concept of *barbs*: we write $A \Downarrow a$ if process a is able to send a message on channel a, i.e. $A \rightarrow^* C[\mathsf{out}(a, t).P]$ for some evaluation context $C[\_]$ that does not bind a, some process P and term t.

Definition 5.1. Observational equivalence, denoted $\approx$, is the largest symmetric relation $\mathcal{R}$ on closed extended processes with same domain such that if $A\ \mathcal{R}\ B$ then

1. if $A \Downarrow a$ then $B \Downarrow a$;
2. if $A \to^* A'$ then there exists B' such that $B \to^* B'$ and $A' \mathcal{R} B'$;
3. for all closing evaluation context $C[\_]$ we have that $C[A] \mathcal{R} C[B]$.

Intuitively, the aim of the attacker (modelled by the context $C[\_]$) is to output on channel a whenever he believes that he interacts with A and not with B.

Example 5.3. Consider the following two processes A and B.

$$\begin{array}{rcl} A & = & \mathsf{in}(c,x).\ \mathsf{if}\ x = 0\ \mathsf{then}\ \mathsf{out}(c,1) \\ B & = & \mathsf{in}(c,x).\ \mathsf{if}\ x = 0\ \mathsf{then}\ \mathsf{out}(c,0) \end{array}$$

where 0 and 1 are constants. These two processes are *not* observationally equivalent, i.e., $A \not\approx B$. A witness of the non-equivalence is for instance provided by the context

$$C[\_] = \mathsf{out}(c,1).\ \mathsf{in}(c,y).\ \mathsf{if}\ y = 1\ \mathsf{then}\ \mathsf{out}(a,1) \parallel \_$$

We indeed have that $C[A] \to^* \mathsf{out}(a,1)$ and therefore $C[A] \Downarrow a$ while $C[B]$ can never emit on a.

Example 5.4. As another example consider the following processes A and B

$$\begin{array}{rcl} A & = & \mathsf{in}(c,x).\ \nu n.\ \mathsf{out}(c,h(n)) \\ B & = & \mathsf{in}(c,x).\ \nu n.\ \mathsf{out}(c,h(\langle x,n\rangle)) \end{array}$$

where h is a free symbol, i.e., not appearing in the equational theory. One may think of h as modeling a hash function. These two processes are observationally equivalent, i.e. $A \approx B$ even though this is not straightforward to prove.

5.3.3 Labelled semantics and labelled bisimulation

In the previous example we presented two processes A and B that are observationally equivalent. Showing observational equivalence formally is however tricky as it requires to show that the processes behave in the same way *for all* context $C[\_]$. This universal quantification over

contexts is generally difficult to reason about and motivates the introduction of a labelled semantics, which allows processes to directly interact with the environment.

The *labelled operational semantics* defines the relation $\xrightarrow{\alpha}$ where α is either $\mathsf{in}(a,t)$ (a is a channel name and t is a term that can contain names and variables), or $\nu x.\mathsf{out}(a,x)$ (x is a variable of base type), or $\mathsf{out}(a,c)$ or $\nu c.\mathsf{out}(a,c)$ (c is a channel name). $\xrightarrow{\alpha}$ extends the internal reduction ($\rightarrow$) by the following rules:

$$\textsc{In} \qquad \mathsf{in}(a,x).P \xrightarrow{in(a,t)} P\{^t/_x\}$$

$$\textsc{Out-Ch} \qquad \mathsf{out}(a,c).P \xrightarrow{out(a,c)} P$$

$$\textsc{Open-Ch} \qquad \frac{A \xrightarrow{out(a,c)} A' \qquad c \neq a}{\nu c.A \xrightarrow{\nu c.\mathsf{out}(a,c)} A'}$$

$$\textsc{Out-T} \qquad \mathsf{out}(a,t).P \xrightarrow{\nu x.out(a,x)} P \mid \{^t/_x\} \qquad x \notin \mathsf{fv}(P) \cup \mathsf{fv}(t)$$

$$\textsc{Scope} \qquad \frac{A \xrightarrow{\alpha} A' \qquad u \text{ does not occur in } \alpha}{\nu u.A \xrightarrow{\alpha} \nu u.A'}$$

$$\textsc{Par} \qquad \frac{A \xrightarrow{\alpha} A' \quad bv(\alpha) \cap fv(B) = bn(\alpha) \cap fn(B) = \emptyset}{A \mid B \xrightarrow{\alpha} A' \mid B}$$

$$\textsc{Struct} \qquad \frac{A \equiv B \quad B \xrightarrow{\alpha} B' \quad A' \equiv B'}{A \xrightarrow{\alpha} A'}$$

The rule In allows the environment to input a term and annotates this transition by the label $in(a,t)$. Intuitively, t is the *recipe* (as in §4.2.1) that allows to deduce the term that is input. The fact that recipes do not contain restricted names is enforced by the rule Scope. We distinguish different ways of outputting terms. The distinction is due to the fact that channel names are handled differently from terms of base type to ensure that they do not appear in the frame. The rule Out-Ch simply outputs a public channel name. Open-Ch allows to output a private channel name: this channel name is *opened* and be-

comes public by removing the "νc" specified in the label $\nu c.out(a, c)$. The side-condition $c \neq a$ simply ensures that one cannot open a private channel, by outputting it on itself. The rule OUT-T allows to output terms of base type. Terms are output *by reference*, i.e., an output creates an entry $\{^t/_x\}$ in the frame, allowing the environment to address t through the variable x. The label $\nu x.out(c, x)$ indicates that x is a fresh variable that has been "opened" and can now be used by the environment. SCOPE and PAR are used to close the labelled reduction under evaluation context, provided these contexts do not interfere with the bound names and variables. We provide examples below illustrating the importance of these side-conditions. Finally, the rule STRUCT states that labelled reduction is closed under structural equivalence.

Remark 5.1. Our definition of labelled semantics slightly differs from the definition given in [Abadi and Fournet, 2001]. We prefer this presentation as it clarifies the distinction between channel names and terms of base type. Delaune et al. [2010a] show that observational equivalence coincides for both semantics.

Example 5.5. Let A be the following process

$$A = \nu s.\ \mathsf{out}(c, \mathsf{enc}(s, k)).\ \mathsf{in}(c, y).\ \mathsf{if}\ y = s\ \mathsf{then}\ P$$

where $x, y \notin \mathsf{fv}(P)$. This models a simple challenge-response protocol: the protocol outputs a secret s encrypted with a key k and only proceeds if it receives s. However, the key k has not been declared private. Therefore an attacker can indeed provide s through the following transition sequence.

$$A \xrightarrow{\nu x.out(c,x)} \nu s.\ A_1 \xrightarrow{in(c,\mathsf{dec}(x,k))} \nu s.\ A_2 \to P$$

where

$$\begin{array}{rcl} A_1 & = & \mathsf{in}(c, y).\ \mathsf{if}\ y = s\ \mathsf{then}\ P \parallel \{^{\mathsf{enc}(s,k)}/_x\} \\ A_2 & = & \mathsf{if}\ \mathsf{dec}(x, k) = s\ \mathsf{then}\ P \parallel \{^{\mathsf{enc}(s,k)}/_x\} \end{array}$$

The step $\nu s.\ A_1 \xrightarrow{in(c,\mathsf{dec}(x,k))} \nu s.\ A_2$ can be shown to be a valid transition as follows.

$$\dfrac{\dfrac{\dfrac{}{\mathsf{in}(c,y).\ \mathsf{if}\ y = s\ \mathsf{then}\ P \xrightarrow{in(c,\mathsf{dec}(x,k))} \mathsf{if}\ \mathsf{dec}(x,k) = s\ \mathsf{then}\ P}\ \textsc{In}}{A_1 \xrightarrow{in(c,\mathsf{dec}(x,k))} A_2}\ \textsc{Par}}{\nu s.\ A_1 \xrightarrow{in(c,\mathsf{dec}(x,k))} \nu s.\ A_2}\ \textsc{Scope}$$

We may now consider the process $\nu k.\ A$, which uses a private key k for the challenge-response. Then we have that

$$\nu k.\ A \xrightarrow{\nu x.out(c,x)} \nu k.\ \nu s.\ A_1$$

However the transition

$$\nu k.\ A_1 \xrightarrow{in(c,\mathsf{dec}(x,k))} \nu k.\ \nu s.\ A_2$$

is *not* valid. In particular, the above proof that $\nu s.\ A_1 \xrightarrow{in(c,\mathsf{dec}(x,k))} \nu s.\ A_2$ cannot be extended using the rule Scope: the rule's side condition is violated because k does appear in the label $in(c, \mathsf{dec}(x,k))$. In this way we ensure that the attacker may not directly use restricted names in the terms that are input. In other words, only deducible terms may be sent by the attacker.

As discussed above proving observational equivalence is particularly tricky due to the universal quantification over all contexts. Therefore we now introduce a *labelled bisimulation*.

Definition 5.2. *Labelled bisimilarity*, denoted $\approx_\ell$, is the largest symmetric relation $\mathcal{R}$ on closed extended processes, such that if $A\mathcal{R}B$ then we have

- $A \sim B$;
- if $A \to A'$ then there exists B' such that $B \to^* B'$ and $A'\mathcal{R}B'$;
- if $A \xrightarrow{\alpha} A'$ and $\mathsf{fv}(\alpha) \subseteq \mathsf{Dom}(A)$ and $\mathsf{bn}(\alpha) \cap \mathsf{n}(B) = \emptyset$ then there exists B' such that $B \to^* \xrightarrow{\alpha} \to^* B$ and $A'\mathcal{R}B'$;

The conditions 2 and 3 correspond to the standard definition of (weak) bisimulation. The first condition requires that the processes are statically equivalent, i.e., the attacker cannot distinguish the sequence of terms output so far by the processes.

Abadi and Fournet [2001] show that labelled bisimulation and observational equivalence coincide.

Theorem 5.1 ([Abadi and Fournet, 2001])**.** Let A and B be two closed extended processes. $A \approx B$ if and only if $A \approx_\ell B$.

We can therefore use labelled bisimulation as a proof technique to show that two processes are (or are not) observationally equivalent. Labelled bisimulation indeed avoids the universal quantification over all possible contexts. One may nevertheless notice that the transition system defined by the labelled reduction relation is infinite branching as the input rule may allow an infinite number of possible terms to be sent by the adversary.

We now revisit Examples 5.3 and 5.4.

Example 5.6. Recall the processes

$$\begin{aligned} A &= \mathsf{in}(c,x).\ \mathsf{if}\ x = 0\ \mathsf{then}\ \mathsf{out}(c,1) \\ B &= \mathsf{in}(c,x).\ \mathsf{if}\ x = 0\ \mathsf{then}\ \mathsf{out}(c,0) \end{aligned}$$

defined in Example 5.3. To show that these processes are not observationally equivalent we provided a context that distinguishes them. We provide an alternate way to prove this result using labelled bisimulation.

We proceed by contradiction. Suppose that $A \approx_\ell B$. We have that $A \xrightarrow{in(c,0)} \to \xrightarrow{\nu x.out(c,x)} A'$ where $A' = \{^1/_x\}$. Then, because $A \approx_\ell B$ there must exist B' such that $B \xrightarrow{in(c,0)} \to^* \xrightarrow{\nu x.out(c,x)} \to^* B'$ and $A' \approx_\ell B'$. In the given example the process B', such that $B \xrightarrow{in(c,0} \to^* \xrightarrow{\nu x.out(c,x)} \to^* B'$ is uniquely defined (up to $\equiv$) as $\{^0/_x\}$. However, $A' \not\sim B'$, violating condition 1 of Definition 5.2, hence contradicting that $A \approx_\ell B$.

Example 5.7. In Example 5.4 we defined processes

$$\begin{aligned} A &= \mathsf{in}(c,x).\ \nu n.\ \mathsf{out}(c,h(n)) \\ B &= \mathsf{in}(c,x).\ \nu n.\ \mathsf{out}(c,h(\langle x,n\rangle)) \end{aligned}$$

which we claimed to be observationally equivalent. We now define the relation $\mathcal{R}$ on closed extended processes as

$$\begin{aligned}\mathcal{R} = (A, B) \cup \{(A', B') \mid\ & A' \equiv \nu n.\ \mathsf{out}(c, h(n)),\\ & B' \equiv \nu n.\ \mathsf{out}(c, h(\langle t, n\rangle)),\\ & t \text{ ground}\}\\ \cup \{(A', B') \mid\ & A' \equiv \nu n.\ \{^{h(n)}/_{x}\}\\ & B' \equiv \nu n.\ \{^{h(\langle t,n\rangle)}/_{x}\}\\ & t \text{ ground}, x \in \mathcal{X}\}\end{aligned}$$

Provided that for any ground term t we have that

$$\nu n.\ \{^{h(n)}/_{x}\} \sim \nu n.\ \{^{h(\langle t,n\rangle)}/_{x}\}$$

it is easy to verify that $\mathcal{R}$ satisfies the 3 conditions of Definition 5.2. Hence, as $A\ \mathcal{R}\ B$ we have that $A \approx_\ell B$. We see that labelled bisimulation can be used to reduce the proof of observational equivalence to a proof of a (generally infinite) family of static equivalences—in this example the static equivalence must hold for all ground terms t.

5.4 Exercises

Exercise 13 (**). Consider the previously introduced process P^6_{nspk} which models the Needham-Schroeder public key protocol.

- Give the labelled transition sequence that witnesses Lowe's man in the middle attack.
- Provide the attacker context that allows to witness this attack using only internal reduction.

Exercise 14 (***). The following message exchange describes the Woo Lam mutual authentication protocol which allows to establish a fresh symmetric key using a trusted server S.

1. $A \rightarrow B: \; A, N_a$
2. $B \rightarrow A: \; B, N_b$
3. $A \rightarrow B: \; \{A, B, N_a, N_b\}_{K_{as}}$
4. $B \rightarrow S: \; \{A, B, N_a, N_b\}_{K_{as}}, \{A, B, N_a, N_b\}_{K_{bs}}$
5. $S \rightarrow B: \; \{B, N_a, N_b, K_{ab}\}_{K_{as}}, \{A, N_a, N_b, K_{ab}\}_{K_{bs}}$
6. $B \rightarrow A: \; \{B, N_a, N_b, K_{ab}\}_{K_{as}}, \{N_a, N_b\}_{K_{ab}}$
7. $A \rightarrow B: \; \{N_b\}_{K_{ab}}$

For any agent C we assume K_{cs} to be a symmetric key whose value is initially known only by agent C and the server S.

- Provide a reasonable and general model of this protocol in the applied pi calculus.
- Show that the protocol admits an attack: the intruder I makes B believe that he is executing the protocol with A (while A did not send any message).

  Hints:

  - *The attack requires to interleave 2 sessions of the B role (and none for A and the server).*
  - *Some ciphertexts may be replaced by random values.*
  - *We suppose that A cannot distinguish the session key K_{ab} from a nonce.*

- Show that your model of the protocol includes this attack by providing a labelled reduction sequence as a witness.

6

Security properties

In this section we discuss how several, important security properties can be modelled. Before defining the security properties we introduce the notion of events. Events are used to annotate processes and are useful to define properties about the executions. Then we discuss reachability properties: (a weak version of) confidentiality, or secrecy, and different flavors of authentication. Next we show how equivalence properties may be used to express a variety of security properties, including strong notions of secrecy, privacy preserving properties, as well as more general properties expressed as indistinguishability from an ideal system.

6.1 Events

We first enrich the process calculus with *events*. Events are simply annotations to make statements about which part of the protocol has been reached. For this we add an event construct to the grammar of plain processes:

$$\mathsf{event}\ e(t).P$$

where e is taken from the set of events, a set of unary function symbols, distinct from the term algebra signature. t is a term used to provide

parameters to the event. We could easily consider events of arbitrary arity, but for notational simplicity we consider only unary ones in the definitions. Multiple parameters can be encoded using pairs in the equational theory. Similarly an event of arity 0 can be simply given a constant for t. In what follows, by abuse of notation, we may use events with an arbitrary number of parameters, relying on this encoding.

As events are merely annotations they should not interfere with the process execution. We therefore simply extend the internal reduction by the rule

$$\textsc{Event} \quad \mathsf{event}\ e(t).\ P \to P$$

We also define what it means for an event to occur.

Definition 6.1. A reduction sequence $R = A_0 \xrightarrow{(\alpha_1)} A_1 \ldots \xrightarrow{(\alpha_n)} A_n$ satisfies the event $e(t)$ at step i if and only if $A_{i-1} \equiv C[\mathsf{event}\ e(t).P]$ and $A_i \equiv C[P]$ for some $C[\_], P$.

We say that R satisfies the event $e(t)$, if there exists i such that R satisfies the event $e(t)$ at step i.

6.2 Secrecy

In many approaches for symbolic protocol analysis secrecy is expressed as a reachability property: the adversary may not reach a state where he knows a secret s, i.e., where the secret s is deducible from the current adversary knowledge. This form of secrecy is sometimes called weak secrecy. We discuss later in this chapter stronger flavors of secrecy expressed as particular equivalence properties.

A first remark is that given a process P, the secrecy of term t is actually not well defined, as illustrated by the following example.

Example 6.1. Consider the process $P = P_1 \parallel P_2 \parallel P_3$ where

$$\begin{array}{lcl} P_1 & = & \nu n.\mathsf{out}(c, n) \\ P_2 & = & \nu n.\mathsf{out}(c, h(n)) \\ P_3 & = & \mathsf{out}(c, h(n)) \end{array}$$

The question of whether n is secret in P is not well defined. Actually, formally, n may only refer to n used in P_3. However, we may wish to

express that the n used in P_1 is secret (which is not the case) and similarly for P_2. However, n is bound in P_1 and in P_2 and may be α-converted at will.

To avoid the problem of being unable to address bound names we add a *monitor process* that raises an event in case the secret is deducible by the adversary.

Definition 6.2. Let A be a closed extended process such that the event ded does not occur in A and $s \in \mathsf{n}(A)$. Let $A^s = \nu s.(A \parallel (in(c,x)$ if $x = s$ then event $ded(s))$. We say that s is (weakly) secret in A if and only if for all reduction sequence $R = A^s \xrightarrow{(\alpha_1)} A_1 \ldots \xrightarrow{(\alpha_n)} A_n$ we have that R does not satisfy $ded(s)$.

Note that for simplicity we restrict this definition to the secrecy of nonces that are restricted at the top level, i.e. the "νs" is placed in front of the process. This definition can be generalized to compound terms as long as all bound names appearing in the term are restricted at the top level. When restrictions appear under replication a general definition is more complicated but in many particular cases the corresponding monitor process is straightforward to add.

We may sometimes abuse language and talk about the secrecy of a restricted name to mean the secrecy of free name in the process where the restriction of this name has been removed. We will only do so when this is not ambiguous, i.e. the name is bound once and does not appear freely.

Example 6.2. Consider again the Needham Schroeder public key protocol and its modelling by the process P^6_{nspk} on p. 38. One may want to check whether the nonces n_a and n_b are secret or not. To avoid referring to a nonce under a replication we may note that by unrolling some replications we have that

$$\begin{aligned} P^6_{\mathsf{nspk}} \quad \equiv \quad & \nu sk.(\mathsf{in}(c, x_{pk}).P'_A(sk, x_{pk}) \parallel \\ & \qquad !\mathsf{in}(c, x_{pk}).P_A(sk, x_{pk}) \parallel !P_B(sk) \parallel \mathsf{out}(c, \mathsf{pk}(sk))) \parallel \\ & P^6_{\mathsf{nspk}} \end{aligned}$$

where $P'_A(sk, x_{pk})$ is obtained from $P_A(sk, x_{pk})$ α-converting n_a to n'_a. We may regard $P'_A(sk, x_{pk})$ as a test session, which has been chosen

without loss of generality, as it was obtained through structural equivalence.

We denote by A the process obtained from the above process by removing the restriction on n'_a. We will also suppose that B is a similar construction, but defining a test session for the responder, which allows us to reason about the secrecy of n'_b.

One might expect that n'_a is secret in A while n'_b is not secret in B due to the man-in-the-middle attack. However, as stated, both nonces may be divulged to the attacker. Indeed if a starts a session with the attacker, the nonce n'_a is trivially leaked.

More formally, we have that the reduction sequence

$$\begin{aligned}
A^{n'_a} = \;& \nu n'_a.\ (A \parallel (in(c,x) \text{ if } x = s \text{ then event } ded(s)) \\
\xrightarrow{\mathsf{in}(c,\mathsf{pk}(sk_c))} & \\
\xrightarrow{\nu x.\mathsf{out}(c,x)} \;& \nu n'_a.\nu sk.(A' \parallel \\
& \qquad (in(c,x) \text{ if } x = s \text{ then event } ded(s)) \parallel \\
& \qquad \{^{\mathsf{aenc}(\langle \mathsf{pk}(sk), n_a\rangle, \mathsf{pk}(sk_c))}/_{x}\}) \\
\xrightarrow{\mathsf{in}(c,\mathsf{snd}(\mathsf{adec}(x,sk_c)))} & \\
\to \;& \nu n'_a.\nu sk.(A' \parallel \text{event } ded(s) \parallel \\
& \qquad \{^{\mathsf{aenc}(\langle \mathsf{pk}(sk), n_a\rangle, \mathsf{pk}(sk_c))}/_{x}\}) \\
\to \;& \nu n'_a.\nu sk.(A' \parallel \{^{\mathsf{aenc}(\langle \mathsf{pk}(sk), n_a\rangle, \mathsf{pk}(sk_c))}/_{x}\})
\end{aligned}$$

witnesses the attack on the secrecy of n'_a.

This example shows that one has to be careful when stating the security properties to be checked. One way to avoid the nonces to be trivially leaked in this way is to enforce the test session between honest participants and define

$$\begin{aligned}
P^{n_a}_{\mathsf{nspk}} \;\hat{=}\; & \nu sk.(P'_A(sk, pk(sk)) \parallel \\
& \qquad !\mathsf{in}(c, x_{pk}).P_A(sk, x_{pk}) \parallel\ !P_B(sk) \parallel \mathsf{out}(c, \mathsf{pk}(sk))) \parallel \\
& P^6_{\mathsf{nspk}}
\end{aligned}$$

We can define similarly P^{n_b} but we have to alter P'_B to additionally include the test

$$\text{if } \mathsf{pk}(sk) = \mathsf{fst}(\mathsf{adec}(y, sk_r)) \text{ then}$$

just before the creation of n'_b ($\nu n'_b$). This results into a process P'_B that only accepts sessions with the honest participants.

Now we indeed have that $P^{n_a}_{\mathsf{nspk}}$ preserves the secrecy of n'_a while $P^{n_b}_{\mathsf{nspk}}$ does not preserve the secrecy of n'_b because of the man-in-the-middle attack.

6.3 Authentication

Authentication is among the most important security properties. One sometimes distinguishes between entity authentication and message authentication. We mostly focus on the former. Roughly, the aim is that an attacker is not able to impersonate an entity A when communicating with an entity B. The ISO/IEC-9798-1 standard defines the goal of (entity) authentication as follows:

> *Entity authentication mechanisms allow the verification, of an entity's claimed identity, by another entity. The authenticity of the entity can be ascertained only for the instance of the authentication exchange.*

This kind of natural language specification leaves however much room for interpretation. The interested reader may refer to Gollmann's discussion [Gollmann, 1996] of different interpretations that may lead to "attacks", depending on which interpretation is chosen. We define below several flavors of authentication, following Lowe's hierarchy of authentication properties [Lowe, 1997a].

6.3.1 Correspondence properties

Woo and Lam [1992] introduced the use of *correspondence properties* to model authentication. Intuitively, a correspondence property states that *if an event e has happened then an event e' must have happened before*. To model authentication the event e is typically a statement of the form *"B accepted a run of the protocol"* and the event e' is a statement of the form *"A started a run of the protocol"*. Blanchet [2009] formalized correspondence properties in the applied pi calculus and proposes a method for automatically verifying these properties in the ProVerif tool. We present here a slightly simplified version of

this formalization which is sufficient to express the different notions of authentication.

Correspondence properties are statements of the form

$$e(t) \rightsquigarrow e'(t')$$

Intuitively, such a property holds on a process A if in any execution of A whenever an instance of the event $e(t)$, say $e(t\sigma)$, occurred, then the corresponding instance of e', i.e. $e'(t'\sigma)$, occurred before. More formally we define the satisfaction of a correspondence property as follows.

Definition 6.3. A closed extended process A satisfies the correspondence property $e(t) \rightsquigarrow e'(t')$ if and only if for all reduction $R = A \xrightarrow{(\alpha_1)} A_1 \ldots \xrightarrow{(\alpha_n)} A_n$ we have that if R satisfies $e(t\sigma)$ for some σ with $\mathsf{Dom}(\sigma) = \mathsf{fv}(t, t')$ then R satisfies $e'(t'\sigma)$.

One may note that in the definition we did not explicitly require that e' occurred before e. This is however an immediate consequence: suppose that the definition holds and that there is a reduction where $e'(t'\sigma)$ only occurs after $e(t\sigma)$; as the definition quantifies over all reductions the smallest prefix of this reduction satisfying $e(t\sigma)$ would violate the definition.

We sometimes need to consider properties where every occurrence of the event e needs to match a different occurrence of e'. This requirement is formalized by the use of *injective* correspondence properties which are of the form

$$e(t) \rightsquigarrow_{\mathsf{inj}} e'(t')$$

and whose satisfaction is defined as follows.

Definition 6.4. A closed extended process A satisfies the injective correspondence property $e(t) \rightsquigarrow_{\mathsf{inj}} e'(t')$ if and only if for all reduction $R = A \xrightarrow{(\alpha_1)} A_1 \ldots \xrightarrow{(\alpha_n)} A_n$, there exists a partial, injective function $\phi : \{1, \ldots, n\} \nrightarrow \{1, \ldots, n\}$ such that if R satisfies $e(t\sigma)$ for some σ with $\mathsf{Dom}(\sigma) = \mathsf{fv}(t, t')$ at step i then R satisfies $e'(t'\sigma)$ at step $\phi(i)$.

Injective correspondence properties protect against replay attacks, as exemplified in the next section.

6.3.2 Authentication as correspondence properties

We review the different flavors of authentication identified in Lowe's authentication hierarchy [Lowe, 1997a] and show how they can be modelled as correspondence properties. For each type of authentication we first recall the natural language definition given by Lowe (up to some minor rewording) and then illustrate how each of these definitions can be formalized as a correspondence property on the Needham Schroeder public key protocol.

Throughout this section we suppose that the processes P_A and P_B, introduced before for modeling the initiator and responder roles of the Needham Schroeder public key protocol are annotated using events $e^a_{\mathsf{begin}}(t^a_{\mathsf{begin}})$, $e^a_{\mathsf{accept}}(t^a_{\mathsf{accept}})$, $e^b_{\mathsf{begin}}(t^b_{\mathsf{begin}})$, and $e^b_{\mathsf{accept}}(t^b_{\mathsf{accept}})$ that are placed as follows.

$$
\begin{array}{rcl}
P_A(sk_i, pk_r) & \hat{=} & \nu n_a.\ \mathsf{out}(c, \mathsf{aenc}(\langle \mathsf{pk}(sk_i), n_a\rangle, \mathsf{pk}_r)).\\
& & \mathsf{event}\ e^a_{\mathsf{begin}}(t^a_{\mathsf{begin}})\\
& & \mathsf{in}(c, x).\\
& & \mathsf{if}\ \mathsf{fst}(\mathsf{adec}(x, sk_i)) = n_a\ \mathsf{then}\\
& & \mathsf{let}\ x_{nb} = \mathsf{snd}(\mathsf{adec}(x, sk_i))\ \mathsf{in}\\
& & \mathsf{event}\ e^a_{\mathsf{accept}}(t^a_{\mathsf{accept}})\\
& & \mathsf{out}(c, \mathsf{aenc}(x_{nb}, \mathsf{pk}_r))
\end{array}
$$

$$
\begin{array}{rcl}
P_B(sk_r) & \hat{=} & \mathsf{in}(c, y).\\
& & \mathsf{let}\ \mathsf{pk}_i = \mathsf{fst}(\mathsf{adec}(y, sk_r))\ \mathsf{in}\\
& & \mathsf{let}\ y_{na} = \mathsf{snd}(\mathsf{adec}(y, sk_r))\ \mathsf{in}\\
& & \nu n_b.\mathsf{event}\ e^b_{\mathsf{begin}}(t^b_{\mathsf{begin}})\\
& & \mathsf{out}(c, \mathsf{aenc}(\langle y_{na}, n_b\rangle, \mathsf{pk}_i))\\
& & \mathsf{in}(c, z).\\
& & \mathsf{if}\ \mathsf{adec}(z, sk_r) = n_b\ \mathsf{then}\\
& & \mathsf{event}\ e^b_{\mathsf{accept}}(t^b_{\mathsf{accept}}).\ Q
\end{array}
$$

Note that the terms $t^a_{\mathsf{begin}}, t^b_{\mathsf{begin}}, t^a_{\mathsf{accept}}, t^b_{\mathsf{accept}}$ are left unspecified for the moment. They will be instantiated according to which flavor of authentication we wish to express.

We now review several variants of authentication properties.

Aliveness The weakest form of authentication is aliveness.

Definition 6.5 (Aliveness [Lowe, 1997a])**.** A protocol satisfies aliveness if, whenever an honest agent A completes a run of the protocol, apparently with another honest agent B, then B has previously run the protocol.

To specify aliveness for the initiator we let $t^a_{\mathsf{accept}} = \mathsf{pk}_r$ and $t^b_{\mathsf{begin}} = \mathsf{pk}(sk_r)$. The process P^5_{nspk} guarantees aliveness to the initiator if the correspondence property

$$e^a_{\mathsf{accept}}(\mathsf{pk}(sk_b)) \rightsquigarrow e^b_{\mathsf{begin}}(\mathsf{pk}(sk_b))$$

is satisfied. This correspondence property indeed expresses that whenever the initiator successfully finishes his protocol with b ($e^a_{\mathsf{accept}}(\mathsf{pk}(sk_b))$ is satisfied) then b previously initiated a run of the protocol ($e^b_{\mathsf{begin}}(\mathsf{pk}(sk_b))$ is satisfied). This property is indeed satisfied by P^6_{nspk}.

We note that the more general correspondence property

$$e^a_{\mathsf{accept}}(x) \rightsquigarrow e^b_{\mathsf{begin}}(x)$$

would be trivially violated, as the attacker could act as the responder and would not execute the event e^b_{begin}, i.e., we can only expect authentication to hold between honest agents. This is similar to the use of test sessions when specifying secrecy.

As the Needham Schroeder protocol is designed to guarantee *mutual* authentication we can also express that the protocol guarantees aliveness to the responder by defining $t^b_{\mathsf{accept}} = \mathsf{pk}_i$, $t^a_{\mathsf{accept}} = \mathsf{pk}(sk_i)$ and requiring

$$e^b_{\mathsf{accept}}(\mathsf{pk}(sk_a)) \rightsquigarrow e^a_{\mathsf{begin}}(\mathsf{pk}(sk_a))$$

to hold. Again this property is satisfied by P^5_{nspk}. The fact that the property holds illustrates the weak nature of aliveness. It does not capture the man in the middle attack: indeed aliveness simply requires a to be alive, which is indeed the case in case of the man in the middle attack.

Weak agreement As we have seen aliveness fails to capture some attacks. We therefore define the notion of agreement, stating that the initiator and responder should agree on their identities.

Definition 6.6 (Weak agreement [Lowe, 1997a])**.** A protocol guarantees weak agreement if, whenever an honest agent A completes a run of the protocol, apparently with another honest agent B, then B has previously been running the protocol, apparently with A.

Weak agreement for the initiator is expressed by including both identities as parameters to the events: we define $t^a_{\mathsf{accept}} = \langle \mathsf{pk}(sk_i), \mathsf{pk}_r \rangle$ and $t^b_{\mathsf{begin}} = \langle pk_i, \mathsf{pk}(sk_r) \rangle$ and require the property

$$e^a_{\mathsf{accept}}(\langle \mathsf{pk}(sk_a), \mathsf{pk}(sk_b) \rangle) \rightsquigarrow e^b_{\mathsf{begin}}(\langle \mathsf{pk}(sk_a), \mathsf{pk}(sk_b) \rangle)$$

to hold.

Similarly, to express weak agreement for the responder we let $t^b_{\mathsf{accept}} = \langle \mathsf{pk}_i, \mathsf{pk}(sk_r) \rangle$ and $t^a_{\mathsf{begin}} = \langle \mathsf{pk}(sk_i), pk_r \rangle$ and require that

$$e^b_{\mathsf{accept}}(\langle \mathsf{pk}(sk_a), \mathsf{pk}(sk_b) \rangle) \rightsquigarrow e^a_{\mathsf{begin}}(\langle \mathsf{pk}(sk_a), \mathsf{pk}(sk_b) \rangle)$$

The process P^5_{nspk} satisfies weak agreement to the initiator, but not to the responder: in the man in the middle attack the responder accepts a run between a and b while the initiator started a run between a and the attacker. We see that this notion of authentication successfully captures this attack.

Non-injective agreement Sometimes, it is not sufficient to authenticate the other user's identity. We may also want to ensure that both parties agree on some other messages, e.g. the nonces n_a and n_b in the Needham Schroeder public key protocol. Therefore we may want to parametrize the definition by a set of data which the parties should agree on.

Definition 6.7 (Non-injective agreement [Lowe, 1997a])**.** A protocol guarantees non-injective agreement if, whenever an honest agent A completes a run of the protocol, apparently with another honest agent B on a set of data d then B has previously been running the protocol, apparently with A with the same set of data d.

Generally one expects the participants to agree on all atomic data. This property is sometimes called full agreement, or just agreement without specifying d. Agreement for the initiator could be expressed by defining $t^a_{\mathsf{accept}} = \langle \mathsf{pk}(sk_i), \langle \mathsf{pk}_r, \langle n_a, x_{nb} \rangle \rangle \rangle$ and $t^b_{\mathsf{begin}} = \langle pk_i, \langle \mathsf{pk}(sk_r), \langle y_{na}, n_b \rangle \rangle$ and requiring the property

$$e^a_{\mathsf{accept}}(\langle \mathsf{pk}(sk_a), \langle \mathsf{pk}(sk_b), \langle x, y \rangle \rangle \rangle) \rightsquigarrow e^b_{\mathsf{begin}}(\langle (\langle \mathsf{pk}(sk_a), \langle \mathsf{pk}(sk_b), \langle x, y \rangle \rangle \rangle)$$

to hold. Agreement for the responder is modelled in a similar way.

Injective agreement Non-injective agreement may still not be sufficiently strong. Consider the following protocol for sharing a symmetric key:

$$A \to B : \quad \mathsf{sign}(\mathsf{aenc}(k, \mathsf{pk}(B)), \mathsf{prv}(A))$$

A generates a fresh key k, encrypts it with B's public encryption key and signs the encryption with his own private signing key.

This protocol should guarantee full (non-injective) agreement, i.e., both A and B agree on the identities and the key. However, an attacker may *replay* A's message and trick B into using a previously used key. This key might however have been compromised since its first usage, maybe because of a long brute force attack, or careless disposal of previous key material. Therefore, one may want to ensure that such replay attacks are not possible, which motivates the notion of *injective* agreement.

Definition 6.8 (Injective agreement [Lowe, 1997a])**.** A protocol guarantees injective agreement if, whenever an honest agent A completes a run of the protocol, apparently with another honest agent B on a set of data d then B has previously been running the protocol, apparently with A with the same set of data d and each such run of A corresponds to a *unique* run of B.

Injective agreement is modelled as non-injective agreement, but we replace $\rightsquigarrow$ with $\rightsquigarrow_{\mathsf{inj}}$. The above described protocol would obviously violate this property as the attacker could replay A's message resulting

into several identical accept events for a single begin event. This kind of replay attacks is avoided in the (Lowe-)Needham Schroeder public key protocol by the nonce handshake, which guarantees freshness of the messages.

6.4 Equivalence properties

6.4.1 Strong flavors of confidentiality

Expressing confidentiality properties in terms of deduction is often too weak. Indeed a term is secret unless the complete term may be known to the adversary. Consider for instance a function h that when applied to a pair returns the first element of the pair, modelled by the equation $h(\langle x, y\rangle) = x$. Given $h(t)$, the term t would be declared secret according to our previous definition. If the hash would be applied to your credit card number, modelled as the nested pairs of each of the digits, this would certainly not be an acceptable notion of security, as the adversary would learn all but one digit.

This is why Blanchet introduced the notion of *strong secrecy* [Blanchet, 2004], which is inspired by the definitions of *semantic security* in cryptography. Strong secrecy requires that two executions of the protocol are indistinguishable, even if the adversary may choose the value of the secret in each of the two runs.

Definition 6.9 (strong secrecy)**.** Let P be an extended process and $\mathsf{fv}(P) = x$. We say that x is strongly secret in P if and only if

$$\mathsf{in}(c, x_1).\ \mathsf{in}(c, x_2).P\{^{x_1}/_{x}\} \approx \mathsf{in}(c, x_1).\ \mathsf{in}(c, x_2).P\{^{x_2}/_{x}\}$$

The fact that the adversary choses the secret value is expressed by the input of two possible values as a preamble to the protocol.

Example 6.3. Consider the protocol A that simply outputs the public key encryption of a secret.

$$A = \nu sk.\ \mathsf{out}(c, \mathsf{aenc}(x, \mathsf{pk}(sk))) \parallel \{^{\mathsf{pk}(sk)}/_{y}\}$$

The active substitution models that the public key is known to the adversary. It is easy to see that x is not strongly secret in A, i.e.,

$$\mathsf{in}(c, x_1).\ \mathsf{in}(c, x_2).A\{^{x_1}/_{x}\} \not\approx \mathsf{in}(c, x_1).\ \mathsf{in}(c, x_2).A\{^{x_2}/_{x}\}$$

Indeed, it is sufficient for the attacker to provide two constants c_1, and c_2 as input values and compare the output with $\mathsf{aenc}(c_1, y)$, the encryption of c_1 with the public key. One may note that the process $\nu s.A\{^{s}/_{x}\}$ does preserve the weak secrecy of s.

This example also illustrates that our modeling of encryption is too naive, as it does not reflect the randomness of (most) asymmetric encryption schemes. While this abstract view of encryption is generally sufficient for trace properties it is no longer the case when considering indistinguishability properties. If we consider an updated model where aenc is a ternary function with an explicit argument for the randomness (and the expected, updated equational theory) we indeed have that

$$A = \nu sk.\ \nu r.\mathsf{out}(c, \mathsf{aenc}(x, r, \mathsf{pk}(sk))) \parallel \{^{\mathsf{pk}(sk)}/_{y}\}$$

A satisfies strong secrecy of x. This property can be easily checked using a tool such as ProVerif.

Another flavor of secrecy of interest is *real-or-random* secrecy. In a first phase, the adversary is allowed to interact with the protocol at will. In a second phase the adversary is given either the real secret, or a fresh randomly generated secret. If the adversary is unable to distinguish these two scenarios, we say that the protocol satisfies real-or-random secrecy.

Definition 6.10 (real-or-random secrecy)**.** Given a closed extended process A, such that $s \notin \mathsf{bn}(A)$ we say that $\nu s.A$ satisfies real-or-random secrecy of the name (of base type) s if and only if for all B and reduction sequence $\nu s.\ A \xrightarrow{\alpha_1} \nu s.\ A_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} \nu s.\ A_n$ such that $s \notin \mathsf{bn}(A_i)$ for $1 \leq i \leq n$ and no α-conversion in this reduction sequence involved s we have that

$$\nu s.(\phi(A_n) \parallel \{^{s}/_{x}\}) \sim \nu s.(\phi(A_n) \parallel \nu s'.\ \{^{s'}/_{x}\})$$

Note that we need to suppose that no α-conversion involved the value s in the reductions in order to make sure we still refer to the right name. One may note that real-or-random secrecy is strictly weaker than strong secrecy.

Proposition 6.1. Let A be an extended process, such that $\mathsf{fv}(A) = \{x\}$, x is of base type and $s \notin \mathsf{bn}(A) \cup \mathsf{n}(A)$. If x is strongly secret in A then $\nu s.A\{^{s}/_{x}\}$ satisfies real-or-random secrecy of s. The converse is not true in general.

The proof of this proposition relies on the following two lemmas. The first lemma states that any transition under a restriction is also valid without the restriction.

Lemma 6.1. Let s be a name of base type. If $\nu s.A \xrightarrow{\alpha} \nu s.B$ (where α may be empty, denoting an internal reduction) and $s \notin \mathsf{bn}(A) \cup \mathsf{bn}(B)$ and no α-conversion in this sequence involved s then $A \xrightarrow{\alpha} B$.

Proof. The proof is done by induction on the proof tree of $\nu s.A \xrightarrow{\alpha} \nu s.B$. □

The second lemma is a result on static equivalence.

Lemma 6.2. For any frames φ, φ' we have that if $\varphi \sim \varphi'\{^{s'}/_{s}\}$ then $\nu s.(\varphi \mid \{^{s}/_{x}\}) \sim \nu s.\varphi \mid \nu s'.\{^{s'}/_{x}\}$.

Proof. We have that

$$
\begin{array}{lll}
 & \varphi \sim \varphi'\{^{s'}/_{s}\} & \\
\Rightarrow & \nu s.\varphi \sim \nu s.\varphi'\{^{s'}/_{s}\} & \text{by Lemma 4.4} \\
\Leftrightarrow & \nu s.\varphi \sim \varphi'\{^{s'}/_{s}\} & \text{as } s \text{ does not occur in } \varphi'\{^{s'}/_{s}\} \\
\Leftrightarrow & \nu s.\varphi \sim \varphi & \text{as, by hyp., } \varphi \sim \varphi'\{^{s'}/_{s}\} \\
\Leftrightarrow & \nu s'.\varphi\{^{s'}/_{s}\} \sim \varphi & \text{by } \alpha\text{-conversion} \\
\Rightarrow & \nu s.\nu s'.(\varphi\{^{s'}/_{s}\} \mid \{^{s}/_{x}\}) \sim \nu s.(\varphi \mid \{^{s}/_{x}\}) & \\
 & & \text{by Lemma 4.4} \\
\Leftrightarrow & \nu s'.\varphi\{^{s'}/_{s}\} \mid \nu s.\{^{s}/_{x}\} \sim \nu s.(\varphi \mid \{^{s}/_{x}\}) & \\
 & & \text{as } s \text{ does not occur in } \varphi\{^{s'}/_{s}\} \\
\Leftrightarrow & \nu s.\varphi \mid \nu s'.\{^{s'}/_{x}\} \sim \nu s.(\varphi \mid \{^{s}/_{x}\}) & \text{by } \alpha\text{-conversion}
\end{array}
$$

□

We can now prove Proposition 6.1

Proof. We will prove the contraposition. Suppose that $\nu s.A\{^{s}/_{x}\}$ does not satisfy real-or-random secrecy of s: there exists $A_1, \ldots, A_n$ such

that

$$\nu s.A\{^{s}/_{x}\} \xrightarrow{\alpha_1} \nu s.A_1 \ldots \xrightarrow{\alpha_n} \nu s.A_n$$

and

$$\nu s.\phi(A_n) \mid \{^{s}/_{x}\} \not\approx \nu s.\phi(A_n) \mid \nu s'.\{^{s'}/_{x}\} \tag{6.1}$$

Note that s may not occur in any label α_i as s is a restricted name of base type. We have that

$$\mathsf{in}(c, x_1).\ \mathsf{in}(c, x_2).\ A\{^{x_1}/_{x}\} \xrightarrow{in(c,s)} \xrightarrow{in(c,s')} A\{^{s}/_{x}\}$$

and by Lemma 6.1 we also have that

$$A\{^{s}/_{x}\} \xrightarrow{\alpha_1} A_1 \ldots \xrightarrow{\alpha_n} A_n$$

Moreover, for any A' such that

$$\mathsf{in}(c, x_1).\ \mathsf{in}(c, x_2).\ A\{^{x_2}/_{x}\} \xrightarrow{in(c,s)} \xrightarrow{in(c,s')} \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} A'$$

we have that s does not occur in A' as s does not occur in A nor in any label α_i and A inputs on x_2 (therefore inputs s' and not s). Hence, $\phi(A') = \phi(A')\{^{s'}/_{s}\}$. Applying Lemma 6.2 to the static inequivalence (6.1) with $\varphi = \phi(A_n)$ and $\varphi' = \phi(A')$ we have that $\phi(A_n) \not\approx \phi(A')\{^{s'}/_{s}\}$. As $\phi(A') = \phi(A')\{^{s'}/_{s}\}$, we conclude

$$\mathsf{in}(c, x_1).\ \mathsf{in}(c, x_2).\ A\{^{x_1}/_{x}\} \not\approx_{\ell} \mathsf{in}(c, x_1).\ \mathsf{in}(c, x_2).\ A\{^{x_2}/_{x}\}$$

To see that the converse direction of the proposition does not hold consider the following counter-example. Let c_1 and c_2 be two constants and

$$A = \mathsf{in}(c, y).\ \mathsf{if}\ y = x\ \mathsf{then}\ \mathsf{out}(c, c_1)\ \mathsf{else}\ \mathsf{in}(c, c_2)$$

The process A does not preserve strong secrecy of x as the adversary may force execution of the then branch on one side and the else branch on the other side. However, the process A does preserve real-or-random secrecy of x. □

Real-or-random secrecy is also useful to reason about password based protocols. Password based protocols may be subject to offline dictionary attacks. Offline dictionary attacks consist of two phases: in the first phase an attacker interacts with a protocol and collects some data; in the second phase the adversary tries all possible passwords in a dictionary and uses the collected data to verify whether the current entry is the correct one or not. When a protocol satisfies real-or-random secrecy of a password, it resists against offline dictionary attacks. Indeed the first phase of an offline dictionary attack is captured in Definition 6.10 by universally quantifying over B. The second phase is captured by the static equivalence

$$\nu s.(\phi(B) \parallel \{^{s}/_{x}\}) \sim \nu s.(\phi(B) \parallel \nu s'.\ \{^{s'}/_{x}\})$$

Intuitively, the variable x can be interpreted as the dictionary entry the adversary is currently checking and the equivalence can be read as "trying the *right* entry in the dictionary is indistinguishable from another entry".

First definitions for offline dictionary attacks were proposed by Lowe [2004] and Delaune and Jacquemard [2006]. Modeling offline dictionary attacks using static equivalence was first proposed by Corin et al. [2005]. Automated verification of real-or-random secrecy was shown decidable by Baudet [2005] for a bounded number of sessions and its verification is also supported by the ProVerif tool for an unbounded number of sessions [Blanchet, 2004].

6.4.2 Privacy properties

In the literature one may find many examples of privacy properties that can be expressed in terms of process equivalences. These definitions include anonymity (towards external parties) in authentication protocols [Abadi and Fournet, 2004], privacy of e-voting [Delaune et al., 2009b] and e-auction protocols [Dong and Pang, 2011, Dreier et al., 2013], unlinkability in RFID protocols [Arapinis et al., 2010, Brusò et al., 2010], etc. We do not aim to give a general definition here, as it would result into a vague and very abstract definition. Instead, we provide a few examples.

Private authentication The notion of private authentication was introduced by Abadi and Fournet [2004]. The idea is that A may try to authenticate to B. B only accepts authentication requests from a given set of users, and refuses the authentication request if A is not a member of this set. In addition to authentication this protocol should protect the anonymity of the users trying to connect and must hide whether the authentication request succeeded or not, i.e. hide whether a given identity is in the set or not.

We present here a simplified protocol proposed by Cheval [2012]. We define the following processes:

$$A(sk_a, pk_b) = \nu n_a.\ \mathsf{out}(c, \mathsf{aenc}(\langle n_a, \mathsf{pk}(sk_a)\rangle, pk_b)).\ \mathsf{in}(c, x)$$

$$\begin{aligned} B(sk_b, pk_a) = \nu n_b.\ &\mathsf{in}(c, y).\ \mathsf{let}\ z = \mathsf{adec}(y, sk_b)\ \mathsf{in} \\ &\mathsf{if}\ \mathsf{fst}(z) = \mathsf{pk}(sk_a) \\ &\qquad \mathsf{then}\ \mathsf{out}(c, \mathsf{aenc}(\langle \mathsf{fst}(z), \langle n_b, \mathsf{pk}(sk_b)\rangle\rangle, pk_a)) \\ &\qquad \mathsf{else}\ \mathsf{out}(c, \mathsf{aenc}(n_b, pk_a)) \end{aligned}$$

The process $A(a, b)$ models the initiator with identity a who wishes to authenticate to b. For this he sends a fresh nonce n_a and his public key $\mathsf{pk}(a)$ encrypted with b's public key to b. The process $B(b, a)$ models the responder b willing to get authentication requests from a. If the request contains a's public key b responds with the message $\mathsf{aenc}(\langle \mathsf{fst}(z), \langle n_b, \mathsf{pk}(sk_b)\rangle\rangle, \mathsf{pk}(sk_a))$. Otherwise b sends a *decoy* message $\mathsf{aenc}(n_b, \mathsf{pk}(sk_a))$. This decoy message should be indistinguishable from a valid message to any party except a. In particular, an attacker sending an initial message with a's public key should be unable to know whether a request from a is accepted by b or not. We can formally state this property as follows:

$$\begin{gathered} \nu sk_a, sk_{a'}, sk_b.\ (\varphi \parallel A(sk_a, \mathsf{pk}(sk_b)) \parallel B(sk_b, \mathsf{pk}(sk_a))) \\ \approx \\ \nu sk_a, sk_{a'}, sk_b.\ (\varphi \parallel A(sk_{a'}, \mathsf{pk}(sk_b)) \parallel B(sk_b, \mathsf{pk}(sk_{a'}))) \end{gathered}$$

where $\varphi = \{^{\mathsf{pk}(sk_a)}/_{x_a}, ^{\mathsf{pk}(sk_{a'})}/_{x_{a'}}, ^{\mathsf{pk}(sk_b)}/_{x_b}\}$ ensures that public keys are known to the adversary. The equivalence states that the attacker cannot distinguish the case where a is allowed to authenticate from the case where a' is. One may note that the attacker may spoof requests

from honest parties, as they only require the knowledge of the public keys. For a complete modelling, one should however also consider semi-dishonest sessions where B is willing to answer requests from a dishonest agent. Interestingly, privacy is lost if the attacker checks message length. Indeed, $\mathsf{aenc}(\langle n_a, \langle n_b, \mathsf{pk}(sk_b)\rangle\rangle, \mathsf{pk}(sk_a))$ is likely to be longer than $\mathsf{aenc}(n_b, \mathsf{pk}(sk_a))$. This attack cannot be detected in a standard symbolic model. It is necessary to enhanced the model with length. We refer the reader to [Cheval et al., 2013] for a model and a corrresponding decision procedure for an adversary that compares the length of messages.

Privacy in e-voting protocols In electronic voting, anonymity or vote privacy is a fundamental property. Formally defining vote privacy may however be tricky. Simply changing the identity of a voter would indeed result into distinguishable processes, as in many protocols the identity of the participating voters is revealed, typically to ensure that only eligible voters did vote. Changing the vote also results in two distinguishable processes as the election result is eventually published and the differing vote may yield an observable difference in the result.

One may also make the following observations about vote privacy. In order to achieve privacy in an election one needs to consider at least 2 honest voters. Otherwise, if all dishonest voters collude, it is generally easy for them to deduce the vote of the honest voters. Similarly, one needs to keep in mind that some information is inevitably be leaked through the election outcome: for instance the extreme case of an unanimous vote leaks how each of the voters did vote.

The definition proposed in [Kremer and Ryan, 2005] therefore proposes to consider a voting protocol with two distinguished voters who swap their vote:

$$S[V\{^{a}/_{id},^{v_1}/_{v}\} \parallel V\{^{b}/_{id},^{v_2}/_{v}\}] \approx S[V\{^{a}/_{id},^{v_2}/_{v}\} \parallel V\{^{b}/_{id},^{v_1}/_{v}\}]$$

Here V represents the voter process, and id and v are the variables referring to the identity and the vote. The context S represents the remaining parts of the system, such as the election administrators or other dishonest voters. The definition effectively expresses that it is not possible for an adversary to link a particular voter to a particular vote.

Unlinkability in RFID protocols Yet another kind of privacy property is unlinkability: it is not possible to link several sessions, i.e., infer that the sessions involve a same user. This property is of particular importance in RFID tags to avoid that a person could be traced. Some implementations of the European electronic passport are vulnerable to an attack that allows tracing a person [Chothia and Smirnov, 2010]. Unlinkability in this context can again be expressed as an observational equivalence.

Suppose that the protocol P uses a secret key k which is specific to each tag and the only difference between tags. Then unlinkability can be expressed as

$$!\nu k.\ !P \approx \nu k.\ !P$$

The left-hand side process allows several tags, each of which may execute several sessions. On the right-hand side there is only one tag that may execute several sessions. If these processes are observationally equivalent an attacker cannot distinguish the case where the same tag executes the protocol several times from the case where the protocol may be executed by different tags.

6.4.3 Ideal systems

In software engineering one often starts from a specification and refines it into a concrete implementation. A similar idea may be applied to security protocols. Abadi and Gordon [1999] propose to specify a security protocol by an ideal system, i.e. a protocol which is trivially secure by design and show that the concrete protocol is indistinguishable from its specification. We illustrate this idea in the following example.

Example 6.4. We consider a simple protocol for message authentication P and a corresponding ideal system I.

$$\begin{aligned} P &= \nu k.(\nu r.\mathsf{out}(c, \mathsf{senc}(m, k, r)) \parallel \\ &\quad \mathsf{in}(c, x).\ \mathsf{if}\ \mathsf{valid}(x, k) = \top\ \mathsf{then}\ \mathsf{let}\ y = \mathsf{sdec}(x, k).Q) \\ I &= \nu k.(\nu r.\mathsf{out}(c, \mathsf{senc}(m, k, r)) \parallel \\ &\quad \mathsf{in}(c, x).\ \mathsf{if}\ \mathsf{valid}(x, k) = \top\ \mathsf{then}\ \mathsf{let}\ y = \mathsf{sdec}(x, k).Q\{^m/_y\}) \end{aligned}$$

The protocol assumes that two entities share a key k and the sender outputs the encryption of the message m with k. We suppose that encryption is authentic, i.e., it is infeasible to construct a valid ciphertext without knowing the encryption key. This is modelled by enriching the equational theory for symmetric encryption with the equation

$$\mathsf{valid}(\mathsf{senc}(m, k, r), k) = \top$$

When the recipient receives a valid ciphertext it executes a process Q which may depend on the message. The ideal system I behaves as P, except that it "magically" passes the message m to Q, independently of the received ciphertext.

Authenticity of the message may be expressed by the fact that for any message m, and any process Q, P and I behave in the same way, i.e.

$$\forall m, Q.\ P \approx I$$

Note that in this simple example the property only holds for a single session, as such a simple protocol would allow for replay attacks, in case we would replicate the processes.

One may notice in the above example that the ideal system has been tailored towards the particular protocol. It may be more desirable to have one ideal protocol that can be implemented by several concrete protocols, even if these protocols significantly differ in the way the property is achieved. Simulation based security definitions, which achieve this kind of independence between the property and the specification, have been introduced by Canetti [2001] and Backes et al. [2007] in computational models. These ideas have also been adapted to symbolic models in [Delaune et al., 2009a, Böhl and Unruh, 2013]. We here only sketch the main idea of this approach: one requires the existence of a context S, called a simulator, who is in charge of making the ideal and the concrete protocol look the same, i.e.,

$$\exists S.\ P \approx S[I]$$

The context S is in charge of simulating network communications of the protocol in order to make the processes observationally equivalent.

The simulator must be able to do so without access to any restricted names of the ideal system I (this is syntactically enforced, as S cannot be under the scope of any restriction in I). The rationale behind this approach is that any attack process A on P can be combined with the context S against the ideal system I. As I is correct by construction such an attack cannot exist.

6.5 Exercises

The following exercises require the use of the ProVerif tool available at

`http://proverif.inria.fr`

ProVerif's input language is a variant of the applied pi calculus and property specification is similar to the material discussed in this chapter.

Exercise 15 (**). Model the Needham Schroeder public key protocol in ProVerif and use ProVerif to verify the following properties, respectively find attacks.

1. Verify that the protocol satisfies (weak) secrecy of nonce n_a, but admits an attack on the (weak) secrecy of nonce n_b. (Check that the attack found corresponds to the man in the middle attack)

2. Verify that injective weak agreement is guaranteed to the initiator, while non-injective, weak agreement is violated.

Exercise 16 (**). Correct the model of the previous exercise by applying Lowe's fix (cf Chapter 2). Use ProVerif to verify the following properties.

1. Verify that the protocol satisfies (weak) secrecy for both nonces n_a and n_b.

2. Verify that full injective agreement is guaranteed to both the initiator and the responder.

3. Show that strong secrecy is not guaranteed for the nonce n_a (even if the encryption is probabilistic). Look at ProVerif's output and explain the attack found.

4. Verify that real-or-random secrecy of nonce n_a is preserved.

Exercise 17 (**). Consider the following protocol due to Kao and Chow in 1995.

1. $A \to S$: A, B, Na
2. $S \to B$: $\{A, B, Na, Kab\}_{Kas}, \{A, B, Na, Kab\}_{Kbs}$
3. $B \to A$: $\{A, B, Na, Kab\}_{Kas}, \{Na\}_{Kab}, Nb$
4. $A \to B$: $\{Nb\}_{Kab}$

The protocol involves three roles: the initiator A, the responder B and a trusted server S. Na and Nb are fresh nonces generated by A and B, respectively. Kas and Kbs are long term keys shared between A, respectively B, and the server S. Kab is a fresh session key generated by S.

As usual we suppose that the intruder may take the role of A or B and has a shared key with the server S. Use ProVerif to verify that the protocol guarantees

1. secrecy of Kab;
2. full injective agreement for both A and B.

We now suppose that an attacker may be able to compromise an old session key Kab, i.e., the attacker may record, or interact with a session and obtain the established session key.

Model key compromise of a previous session in ProVerif and show that this leads to an attack, i.e., secrecy of an uncompromised key may be violated. (Compromise of a key may be modelled by simply outputting the compromised key to the adversary.)

7

Automated verification: bounded case

As illustrated in the previous chapters, the design of protocols is error-prone and finding flaws is not an easy task. Therefore, the two last decades have seen the development of decision techniques and corresponding tools to check *automatically* whether a protocol can be attacked. Actually, even a simple property like secrecy is undecidable [Durgin et al., 1999]: no generic tool can check for secrecy without requiring some restrictions on the attacker model. In particular, allowing to spawn an unbounded number of protocol sessions leads to undecidability. We provide a construction showing undecidability in the next chapter, in §8.1.

Therefore many techniques focus on the case where the number of sessions is bounded : assuming that the protocol is executed a limited number of times, can we check whether the secrecy of some data s is preserved? Note that even if the number of sessions is bounded the system to verify is still infinite (due to the infinite number of messages that an attacker may construct and use when interacting with the protocol participants). One of the first decidability results was proposed by Amadio and Lugiez [2000], Amadio et al. [2002]. Rusinowitch and Turuani [2001, 2003] extend this result to a more general framework that includes in

particular composed keys. They also provide a complexity result: secrecy is shown to be (co-)NP-complete. Another algorithm based on constraint systems has been proposed by Millen and Shmatikov [2001] and extended by Comon-Lundh and Shmatikov [2003] to the exclusive or. The algorithm is rather elegant, and amenable to extensions. This is the approach we have chosen to present in this chapter.

7.1 From protocols to constraint systems

Consider again the Needham-Schroeder public key protocol.

$$\begin{array}{lll} 1. & A \to B: & \mathsf{aenc}(\langle a, n_a \rangle, \mathsf{pk}_b) \\ 2. & B \to A: & \mathsf{aenc}(\langle n_a, n_b \rangle, \mathsf{pk}_a) \\ 3. & A \to B: & \mathsf{aenc}(n_b, \mathsf{pk}_b) \end{array}$$

One session of the initiator instantiated by a willing to talk to b can be informally represented by the following two rules.

$$\to \mathsf{aenc}(\langle \mathsf{pk}(sk_a), n_a \rangle, \mathsf{pk}(sk_b)) \tag{7.1}$$

$$\mathsf{aenc}(\langle n_a, x \rangle, \mathsf{pk}(sk_a)) \to \mathsf{aenc}(x, \mathsf{pk}(sk_b)) \tag{7.2}$$

The agent a first sends her identity (represented by her public key $\mathsf{pk}(sk_a)$), together with a fresh nonce n_a, encrypted by the public key of b. Then upon receiving a message of the form $\mathsf{aenc}(\langle n_a, x \rangle, \mathsf{pk}(sk_a))$, the agent a replies by x encrypted by the public key of b. Similarly, one session of the initiator instantiated by a willing to talk to c can be informally represented by the following two rules.

$$\to \mathsf{aenc}(\langle \mathsf{pk}(sk_a), n'_a \rangle, \mathsf{pk}(sk_c)) \tag{7.3}$$

$$\mathsf{aenc}(\langle n'_a, y \rangle, \mathsf{pk}(sk_a)) \to \mathsf{aenc}(y, \mathsf{pk}(sk_c)) \tag{7.4}$$

Finally, one session of the responder, instantiated by b willing to answer to a can be informally represented by a single rule.

$$\mathsf{aenc}(\langle \mathsf{pk}(sk_a), z \rangle, \mathsf{pk}(sk_b)) \to \mathsf{aenc}(\langle z, n_b \rangle, \mathsf{pk}(sk_a)) \tag{7.5}$$

These rules may be triggered in any order, provided 7.1 is played before 7.2 and 7.3 is played before 7.4. Consider for example the following execution order: 7.3, 7.5, followed by 7.4. This is a possible *interleaving*

of the sessions of the protocol. There is an attack if a third party can learn n_b, the nonce generated by b for a. To learn n_b in this scenario, the attacker should be able to:

- build a message of the form $\mathsf{aenc}(\langle \mathsf{pk}(sk_a), z\rangle, \mathsf{pk}(sk_b))$ out of $\mathsf{aenc}(\langle \mathsf{pk}(sk_a), n'_a\rangle, \mathsf{pk}(sk_c))$ and its initial knowledge S_0. This requirement can be represented as follows:

  $$S_0, \mathsf{aenc}(\langle \mathsf{pk}(sk_a), n'_a\rangle, \mathsf{pk}(sk_c)) \Vdash \mathsf{aenc}(\langle \mathsf{pk}(sk_a), z\rangle, \mathsf{pk}(sk_b))$$

  In return, he would learn $\mathsf{aenc}(\langle z, n_b\rangle, \mathsf{pk}(sk_a))$ for some instantiation on z that satisfies the constraint above.

- build a message of the form $\mathsf{aenc}(\langle n'_a, y\rangle, \mathsf{pk}(sk_a))$ out of $\mathsf{aenc}(\langle z, n_b\rangle, \mathsf{pk}(sk_a))$ and the previous messages

  $$S_0, \mathsf{aenc}(\langle \mathsf{pk}(sk_a), n'_a\rangle, \mathsf{pk}(sk_c)), \mathsf{aenc}(\langle z, n_b\rangle, \mathsf{pk}(sk_a)) \Vdash \mathsf{aenc}(\langle n'_a, y\rangle, \mathsf{pk}(sk_a))$$

- deduce n_b

  $$S_0, \mathsf{aenc}(\langle \mathsf{pk}(sk_a), n'_a\rangle, \mathsf{pk}(sk_c)), \mathsf{aenc}(\langle z, n_b\rangle, \mathsf{pk}(sk_a)), \mathsf{aenc}(y, \mathsf{pk}(sk_c)) \Vdash n_b$$

The attacker can learn n_b if there is an instantiation of the variables x, y, and z that satisfies these three constraints. We formally define constraints in the next section. Note that for simplicity, in the above example, and in the remaining of the chapter we write S, t instead of $S \cup \{t\}$.

Remark 7.1. One may note that in this section we consider a formal model where messages from the attacker are *pattern matched* against the messages (with variables) expected by the protocol. This differs form the approach taken in the applied pi calculus where inputs from the attacker are bound to a variable, and the message is explicitly verified to correspond to the expected message using conditionals. Constraint system approaches that are following the later approach also exist [Delaune and Jacquemard, 2004, Baudet, 2005].

7.1.1 Constraint systems

In the remaining of this chapter, we assume the inference system $\mathcal{I}_{DY}$ defined in Chapter 3. The associated deduction relation is denoted $\vdash$.

Constraints are formally defined as follows.

Definition 7.1. A *constraint* is an expression of the form $S \Vdash u$ where S is a non empty set of terms and u is a term.

A *constraint system* is a set of constraints $C = \bigcup_{i=1}^{n} S_i \Vdash u_i$ such that

1. $S_i \subseteq S_{i+1}$ for any $1 \leq i \leq n-1$;
2. If $S_i \Vdash u_i \in C$ and $x \in \mathsf{var}(S_i)$ then
$$S_j = \min\{S' \mid S' \Vdash v \in C, x \in \mathsf{var}(v)\}$$
is well-defined and is such that $j < i$.

Condition 1 reflects the fact that the knowledge of the attacker increases during the execution (an attacker never forgets). Condition 2 states that variables are introduced in the right of a constraint. When modeling protocols, it will always be the case that variables used in outputs (the left side of constraints) are bound by the input messages (the right side of constraints).

Given some initial knowledge S_0 (a set of terms), a secret data s, and an execution interleaving

$$\begin{array}{rcl} u_1 & \rightarrow & v_1 \\ & \vdots & \\ u_k & \rightarrow & v_k \end{array}$$

the corresponding constraint system is:

$$\begin{array}{rcl} S_0 & \Vdash & u_1 \\ S_0, v_1 & \Vdash & u_2 \\ & \vdots & \\ S_0, v_1, \ldots, v_{k-1} & \Vdash & u_k \\ S_0, v_1, \ldots, v_{k-1}, v_k & \Vdash & s \end{array}$$

The last constraint encodes the security goal, i.e. the secrecy of s.

Example 7.1. We define the constraint system C_{NS} associated to the interleaving of the rules 7.3, 7.5, 7.4 of the Needham-Schroeder protocol as follows.

$$\begin{aligned} S_0, \mathsf{aenc}(\langle \mathsf{pk}(sk_a), n'_a \rangle, \mathsf{pk}(sk_c)) &\Vdash \mathsf{aenc}(\langle \mathsf{pk}(sk_a), z \rangle, \mathsf{pk}(sk_b)) \\ S_1, \mathsf{aenc}(\langle z, n_b \rangle, \mathsf{pk}(sk_a)) &\Vdash \mathsf{aenc}(\langle n'_a, y \rangle, \mathsf{pk}(sk_a)) \\ S_2, \mathsf{aenc}(y, \mathsf{pk}(sk_c)) &\Vdash n_b \end{aligned}$$

where $S_1 = S_0, \mathsf{aenc}(\langle \mathsf{pk}(sk_a), n'_a \rangle, \mathsf{pk}(sk_c))$ and $S_2 = S_1, \mathsf{aenc}(\langle z, n_b \rangle, \mathsf{pk}(sk_a))$.

The set S_0 representing the initial knowledge of the attacker has been left unspecified so far. A possible initial knowledge is:

$$S_0 = \{\mathsf{pk}(sk_a), \mathsf{pk}(sk_b), sk_c\}.$$

The attacker knows the public keys of the honest agents a, b and the private keys of the dishonest agent c.

7.1.2 Solutions

A solution to a constraint system is an instantiation of the variables that satisfy all the constraints. In particular, satisfaction of the last constraint represents the violation of secrecy, while satisfaction of the other constraints ensures the validity of the execution leading to the secrecy violation.

Definition 7.2. A substitution σ is a *solution* to a constraint system C if $S\sigma \vdash u\sigma$ for any constraint $S \Vdash u \in C$.

The notation $\bot$ represents a constrain system that has no solution.

Example 7.2. Consider the constraint system C_{NS} defined in Example 7.1. A solution to C_{NS} is the substitution:

$$\sigma = \{n'_a/z, n_b/y\}.$$

It corresponds the Lowe's man-in-the-middle attack presented in Chapter 2 (Figure 2.1).

7.2 Constraint solving

Searching for attacks against a protocol, for a bounded number of sessions, reduces to searching for the existence of a solution of a constraint system. Indeed, a constraint system describes all possible executions once an interleaving has been fixed and there are finitely many interleavings associated to a finite number of sessions.

We present an algorithm that not only detects whether a constraint system admits a solution but actually computes a finite representation of all possible solutions. This algorithm was developed by Millen and Shmatikov [2001], Comon-Lundh and Shmatikov [2003].

7.2.1 Algorithm

Checking for the existence of a solution is easy when all the right members of the constraints are variables.

$$\begin{array}{rcl} S_0 & \Vdash & x_1 \\ S_0, v_1 & \Vdash & x_2 \\ & \vdots & \\ S_0, v_1, \ldots, v_{k-1} & \Vdash & x_k \end{array}$$

Such a constraint system is said to be in *solved form* and obviously has a solution. Indeed, the substitution σ such that $x_1\sigma = x_2\sigma = x_k\sigma = t_0$ for some $t_0 \in S_0$ is a solution.

Given an arbitrary constraint system, the constraint solving algorithm simplifies the constraints until it gets a system in solved form or a system that is unsatisfiable.

The rule R_{red} detects that a constraint $T \Vdash u$ is redundant. It removes the constraint $T \Vdash u$ in case u is already deducible from the set T increased by variables that appear on the right hand side of solved constraints, that is, variables that must be deducible. The rules R_{unif1} and R_{unif2} guess a possible instantiation of the variables by unifying two non variable subterms of a constraint. R'_{unif2} is a variant of R_{unif2} that does no prohibit the unification of variables in the special case where one of the subterms to be unified is used as the secret key in

$$R_{red} \qquad C \cup \{T \Vdash u\} \rightsquigarrow C$$
$$\text{If } T \cup \{x \mid T' \Vdash x \in C, T' \subseteq T, T' \neq T\} \vdash u$$

$$R_{unif1} \qquad C \cup \{T \Vdash u\} \rightsquigarrow_\sigma C\sigma \cup \{T\sigma \Vdash u\sigma\}$$
$$\sigma = \mathsf{mgu}(t, u), t \in \mathsf{st}(T), t, u \text{ are not variables}$$

$$R_{unif2} \qquad C \cup \{T \Vdash u\} \rightsquigarrow_\sigma C\sigma \cup \{T\sigma \Vdash u\sigma\}$$
$$\sigma = \mathsf{mgu}(t_1, t_2), t_1, t_2 \in \mathsf{st}(T), t_1, t_2 \text{ are not variables}$$

$$R'_{unif2} \qquad C \cup \{T \Vdash u\} \rightsquigarrow_\sigma C\sigma \cup \{T\sigma \Vdash u\sigma\}$$
$$\sigma = \mathsf{mgu}(t_1, t_2), \quad \mathsf{aenc}(t_3, \mathsf{pk}(t_1)), t_2 \in \mathsf{st}(T)$$

$$R_f \qquad C \cup \{T \Vdash f(u, v)\} \rightsquigarrow C \cup \{T \Vdash u, T \Vdash v\}$$
$$f \in \{\mathsf{senc}, \mathsf{aenc}, \mathsf{pair}\}$$

$$R_{unsat} \qquad C \cup \{T \Vdash u\} \rightsquigarrow \bot$$
$$\text{If } var(T) = var(u) = \emptyset \text{ and } T \not\vdash u$$

Figure 7.1: Constraint solving algorithm

a public key encryption. Of course, unification with variables could be also considered. The procedure will still be correct but less efficient. The rule R_f guesses that the attacker has built $f(u, v)$ out of u and v. Finally, R_{unsat} detects that the constraint system is unsatisfiable: there is some ground constraint $T \Vdash u$ such that u is not deducible from T. Since the constraint is ground, it will never be satisfiable.

For the sake of uniformity, we may write $C \rightsquigarrow_\epsilon C'$ instead of $C \rightsquigarrow C'$ where ϵ is the identity substitution.

Example 7.3. Consider again the constraint system C_{NS} defined in Example 7.1.

$$\begin{aligned} S_0, \mathsf{aenc}(\langle \mathsf{pk}(sk_a), n'_a \rangle, \mathsf{pk}(sk_c)) &\Vdash \mathsf{aenc}(\langle \mathsf{pk}(sk_a), z \rangle, \mathsf{pk}(sk_b)) \\ S_1, \mathsf{aenc}(\langle z, n_b \rangle, \mathsf{pk}(sk_a)) &\Vdash \mathsf{aenc}(\langle n'_a, y \rangle, \mathsf{pk}(sk_a)) \\ S_2, \mathsf{aenc}(y, \mathsf{pk}(sk_c)) &\Vdash n_b \end{aligned}$$

After two successive applications of rule R_f with $f = \mathsf{aenc}$ and $f = \mathsf{pair}$, we get that $C_{NS} \rightsquigarrow^2 C^1_{NS}$ where C^1_{NS} is:

$$\begin{aligned} S_0, \mathsf{aenc}(\langle \mathsf{pk}(sk_a), n'_a \rangle, \mathsf{pk}(sk_c)) &\Vdash \mathsf{pk}(sk_b) \\ S_0, \mathsf{aenc}(\langle \mathsf{pk}(sk_a), n'_a \rangle, \mathsf{pk}(sk_c)) &\Vdash \mathsf{pk}(sk_a) \\ S_0, \mathsf{aenc}(\langle \mathsf{pk}(sk_a), n'_a \rangle, \mathsf{pk}(sk_c)) &\Vdash z \\ S_1, \mathsf{aenc}(\langle z, n_b \rangle, \mathsf{pk}(sk_a)) &\Vdash \mathsf{aenc}(\langle n'_a, y \rangle, \mathsf{pk}(sk_a)) \\ S_2, \mathsf{aenc}(y, \mathsf{pk}(sk_c)) &\Vdash n_b \end{aligned}$$

Since $\mathsf{pk}(sk_a), \mathsf{pk}(sk_b) \in S_0$, the two first constraints can be eliminated using the rule R_{red}, yielding C^2_{NS}:

$$\begin{aligned} S_0, \mathsf{aenc}(\langle \mathsf{pk}(sk_a), n'_a \rangle, \mathsf{pk}(sk_c)) &\Vdash z \\ S_1, \mathsf{aenc}(\langle z, n_b \rangle, \mathsf{pk}(sk_a)) &\Vdash \mathsf{aenc}(\langle n'_a, y \rangle, \mathsf{pk}(sk_a)) \\ S_2, \mathsf{aenc}(y, \mathsf{pk}(sk_c)) &\Vdash n_b \end{aligned}$$

We may then apply R_{unif1} to unify $\mathsf{aenc}(\langle z, n_b \rangle, \mathsf{pk}(sk_a))$ and $\mathsf{aenc}(\langle n'_a, y \rangle, \mathsf{pk}(sk_a))$ where $\sigma = \{n_a/z, n_b/y\}$. The resulting constraint system is:

$$\begin{aligned} S_0, \mathsf{aenc}(\langle \mathsf{pk}(sk_a), n'_a \rangle, \mathsf{pk}(sk_c)) &\Vdash n'_a \\ S_1, \mathsf{aenc}(\langle n'_a, n_b \rangle, \mathsf{pk}(sk_a)) &\Vdash \mathsf{aenc}(\langle n'_a, n_b \rangle, \mathsf{pk}(sk_a)) \\ S_2, \mathsf{aenc}(n_b, \mathsf{pk}(sk_c)) &\Vdash n_b \end{aligned}$$

Then the three constraints can be eliminated using the rule R_{red}, yielding the empty (solved) system. Note that σ is precisely the solution we have exhibited in Example 7.2.

7.2.2 Correctness

Given a constraint system, many simplification rules may apply, yielding several possible "simplified" constraint systems. These constraint systems should be further simplified until a solved form is reached or a constraint system is unsatisfiable. The idea is that all paths should be explored, as illustrated in Figure 7.2. As soon as a path leads to a system in solved form then the initial system admits a solution. Conversely, if all paths lead to unsatisfiable systems then the initial system is unsatisfiable.

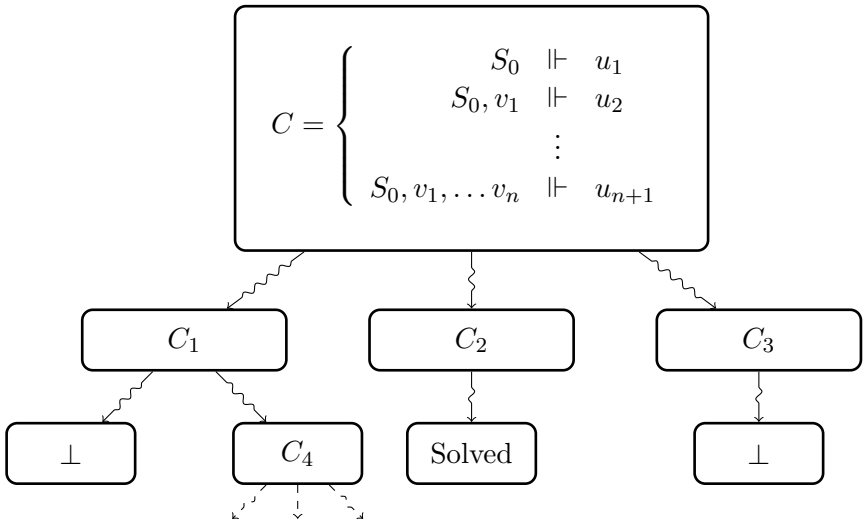

Figure 7.2: Decision procedure.

Formally, this procedure is:

- Sound (Theorem 7.1): any solution found by the procedure is indeed a solution of the constraint system.
- Complete (Theorem 7.2): whenever there is a solution of the constraint system, there is a path in the tree of possible simplifications that leads to a solution.
- Terminating (Theorem 7.3): there is no infinite branch.

The three theorems together actually provide a stronger result: the set of solutions of the initial system can be retrieved by considering the union of the set of solutions of the systems in solved form that appear on the leaves.

Theorem 7.1 (soundness)**.** Let C be a constraint system. If $C \rightsquigarrow_\sigma C'$ then C' is a constraint system and for any solution θ of C', $\sigma\theta$ is a solution of C.

The soundness of the procedure is easy to prove. The most technical point of the proof of Theorem 7.1 lies actually in the proof of Condition 2 of the definition of a constraint system is still satisfied by C'.

Theorem 7.2 (completeness)**.** Let C be a constraint system that is not in solved form. If σ is a solution of C then there exists a system C' and substitutions θ, τ such that $C \rightsquigarrow_\theta C'$ and $\sigma = \theta\tau$.

Theorem 7.2 is of course the most technical result. Its proof can be found in [Millen and Shmatikov, 2001, Comon-Lundh and Shmatikov, 2003, Comon-Lundh et al., 2010] for several variants of the term algebra.

Theorem 7.3 (termination)**.** There is no infinite sequence $C_1 \rightsquigarrow_{\sigma_1} C_2 \rightsquigarrow_{\sigma_2} \cdots$.

The proof of termination is left as an exercise. This procedure actually does not immediately yield an NP-procedure. First, as for all other (NP) procedures, it is necessary to use a DAG representation of the terms. Otherwise, an attack in n steps may require to double the size of the term at each step. Moreover, the length of a branch of the decision procedure is not polynomially bounded as one could expect. While it is rather easy to bound polynomially the DAG-size of each constraint, a constraint may disappear due to application of some rule and re-appear later in the same branch, yielding branches of exponential length, as exemplified by Comon-Lundh et al. [2010]. To obtain a polynomial bound, it is necessary to refine the procedure with strategies, as developed by Comon-Lundh et al. [2010].

7.2.3 Extensions

An enjoyable property of this decision procedure is that no solution is lost. Indeed, a consequence of Theorem 7.2 is that every solution of the initial constraint system can be found in one of the leaves (in solved form) that is reached when applying the procedure. This makes this algorithm more amendable to extensions both in terms of security properties and equational theories. For example, the procedure of

Comon-Lundh et al. [2010] is used to decide a small logic of properties that encompasses, e.g., authentication and also decide whether an attacker can create key cycles on honest keys. Cheval et al. [2011] extended the procedure to cope with equivalence properties, the active counterpart of static equivalence (defined in Chapter 4). It has been implemented yielding the APTE tool [Cheval, 2014]. This procedure has then been used when the attacker is enhanced with the ability to compare the length of messages [Cheval et al., 2013]. Regarding equational theories, [Bursuc et al., 2007] propose a procedure for constraint systems with an associative and commutative function symbol (and no other symbols). Delaune et al. [2012] decide the equivalence of constraint systems for several group theories.

7.2.4 Tools

The first tool to implement such a constraint solving algorithm was presented by Millen and Shmatikov [2001] and further extended by Corin and Etalle [2003], Corin et al. [2006]. The APTE tool [Cheval, 2014] was already mentioned above; it implements a decision procedure based on constraint systems to check equivalence properties.

There are however several other efficient tools, based on different techniques, that can check security properties of protocols for a bounded number of sessions.

- Avispa [Armando et al., 2005] is a plateform that gathers several tools for the analysis of protocols. It currently offers four tools CL-Atse, OFMC, SAT-MC, and TA4SP.

- Scyther [Cremers, 2008] has the originality to allow security analysis for both a bounded and an unbounded number of sessions: the tool first tries to provide a proof of security for an unbounded number of sessions and falls back to the bounded mode in case it fails.

7.3 Exercises

Exercise 18 (**). Prove Theorem 7.3.

Hint: you may use a lexicographical order on (v, s) where v is the number of variables and s the size of the constraint system, for a notion of size to be defined.

Exercise 19 (**). Prove Theorem 7.2.

Exercise 20 (*). Consider the Wide-Mouthed-Frog.

$$\begin{aligned} A &\to S: \quad \{A, B, K_{ab}\}_{K_{as}} \\ S &\to B: \quad \{A, B, K_{ab}\}_{K_{bs}} \end{aligned}$$

A sends to the server a key session K_{ab} using a key K_{as} shared with the server. A also indicates her identity and the identity B of the destination. The server S transmits the session key to B with his shared key K_{bs}. The session key should remain secret between A and B (and S).

Consider a session where the initiator A wishes to talk to B together with a session of the server that answers to A willing to talk to C, some dishonest agent.

1. Show that there is no attack in this configuration, that is, show that the following constraint system has no solution.

$$\begin{aligned} S_0, \{\langle\langle a, b\rangle, k_{ab}\rangle\}_{k_{as}} &\Vdash \{\langle\langle a, c\rangle, x\rangle\}_{k_{as}} \\ S_0, \{\langle\langle a, b\rangle, k_{ab}\rangle\}_{k_{as}}, \{\langle\langle a, c\rangle, x\rangle\}_{k_{cs}} &\Vdash k_{ab} \end{aligned}$$

   where $S_0 = \{a, b, c, k_{cs}\}$.

2. Consider now the following variant of the protocol.

$$\begin{aligned} A &\to S: \quad A, B, \{K_{ab}\}_{K_{as}} \\ S &\to B: \quad A, B, \{K_{ab}\}_{K_{bs}} \end{aligned}$$

   Adapt the constraint system and show that it admits a solution. Exhibit a solution using the constraint solving algorithm.

Exercise 21 (*). We consider the following protocol.

$$\begin{aligned} A &\to B: \quad \{A, K_{ab}\}^{a}_{\mathsf{pk}(B)} \\ B &\to A: \quad \{s\}_{K_{ab}} \end{aligned}$$

1. Show that this protocol admits an attack.
2. Propose a constraint system that reflects the attack scenario.
3. Show, using the constraint solving algorithm that it has a solution.

Exercise 22 (**). We enhance the inference system $\mathcal{I}_{DY}$ defined in Chapter 3 by considering blind signatures (see Exercise 2 for more intuition on blind signatures). Formally, we add the three following inference rules.

$$\frac{x \quad y}{\mathsf{blind}(x,y)} \qquad \frac{\mathsf{sign}(\mathsf{blind}(x,y),z) \quad y}{\mathsf{sign}(x,z)} \qquad \frac{\mathsf{blind}(x,y) \quad y}{x}$$

Accordingly, we add the following simplification rule to the constraint solving algorithm defined in Figure 7.1:

$$C \cup \{T \Vdash \mathsf{blind}(u,v)\} \rightsquigarrow C \cup \{T \Vdash u, T \Vdash v\}$$

1. Show that this rule is correct (easy). More precisely, show that if $C \rightsquigarrow'_C$ with this rule, then any solution θ of C', θ is a solution of C.
2. Show that the resulting constraint solving algorithm is no longer complete. That is, show that Theorem 7.2 is false in this context.

8

Automated verification: unbounded case

In the previous chapter we report on techniques for the analysis of protocols when the number of sessions is bounded. This yields efficient tools to find attacks. However, when no attack is found, it is impossible to conclude whether the analyzed protocol is secure or not. Indeed, there might exist an attack that requires a few additional sessions. Moreover, in practice, tools can only analyse a small number of sessions (typically 2 or 3) in a reasonable amount of time. Therefore, if no attack is found then there is no *proof* that the protocol is secure. To overcome this limitation, it is necessary to analyse protocols for an unbounded number of session. However, even a simple property like secrecy is undecidable when we do not bounded the number of sessions [Durgin et al., 1999]. We provide a construction for undecidability in §8.1.

Nevertheless, it is still possible to design verification tools in this case. Obviously in this case termination is not guaranteed. There have been two main approaches.

- One approach is to perform backward search, relying on causality arguments. This backward search may not terminate, but user interaction with the tool or additional lemmas allow to prune

some branches and enforce termination. Examples of such tools are the NRL Protocol Analyzer [Meadows, 1996], and its reimplementation in Maude, Maude-NPA [Escobar et al., 2009], as well as the Athena [Song, 1999], Scyther [Cremers, 2008] and Tamarin [Schmidt et al., 2012] tools.

- The other approach is to use abstractions. Protocols may be encoded as (first order) Horn clauses [Weidenbach, 1999, Blanchet, 2001] or tree automata [Monniaux, 2003, Goubault-Larrecq, 2000, Genet and Klay, 2000], over-approximating the intruder capabilities. These tools may allow for false attacks and termination is generally still not guaranteed. The ProVerif tool is the most mature tool using this approach: its optimizations and a dedicated Horn clause resolution algorithm make the tool extremely efficient and non-termination and false attacks (at least for weak secrecy properties) rare on practical examples.

We focus on the modeling of protocols as Horn clauses, a modeling first suggested by Weidenbach [1999], and the verification algorithm proposed by Blanchet and implemented in ProVerif [Blanchet, 2001]. The ProVerif tool takes protocols written in a variant of the applied pi calculus as input together with a security property to verify. The protocol is then automatically translated into a set of first-order Horn clauses and the properties are translated into derivability queries. The verification algorithm is based on a dedicated Horn clause resolution procedure. We describe in this chapter how protocols can be encoded as Horn clauses (without providing in full details the automatic translation from the full applied pi calculus). Then we present the first verification algorithm implemented in ProVerif to verify weak secrecy queries. The current version of ProVerif adds many optimizations to this algorithm and allows for verifying more advanced properties (injective and non-injective correspondences and some equivalence based properties) which we do not discuss here.

8.1 Undecidability

Checking for secrecy is (at least) as difficult as checking for a solution to the Post Correspondence Problem (PCP). We show how to encode PCP in protocols.

We first recall the Post Correspondence Problem. Let Σ be a finite alphabet.

Input $(u_i, v_i)_{1 \leq i \leq k}$, $k \in \mathbb{N}$, $u_i, v_i \in \Sigma^*$

Output Does there exist $n \in \mathbb{N}$ and $i_1, \ldots, i_n \in \mathbb{N}$ such that $1 \leq i_j \leq k$ and

$$u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots v_{i_n} \ ?$$

where $\mathbb{N}$ denotes the set of integers.

The Post Correspondence Problem is well known to be undecidable Davis and Weyuker [1983]. Given an input $(u_i, v_i)_{1 \leq i \leq k}$ of PCP, we build the following protocol P.

$$\begin{array}{rcl} & \rightarrow & \{\!|\langle \overline{u_1}, \overline{v_1} \rangle|\!\}^s_{K_{ab}}, \ldots, \{\!|\langle \overline{u_k}, \overline{v_k} \rangle|\!\}^s_{K_{ab}} \\ \{\!|\langle \langle x, y \rangle |\!\}^s_{K_{ab}} & \rightarrow & \{\!|\langle \overline{x \cdot u_1}, \overline{y \cdot v_1} \rangle|\!\}^s_{K_{ab}}, \{\!|s|\!\}^s_{\{\!|\langle \overline{x \cdot u_1}, \overline{x \cdot u_1} \rangle|\!\}^s_{K_{ab}}}, \ldots, \\ & & \{\!|\langle \overline{x \cdot u_k}, \overline{y \cdot v_k} \rangle|\!\}^s_{K_{ab}}, \{\!|s|\!\}^s_{\{\!|\langle \overline{x \cdot u_k}, \overline{x \cdot u_k} \rangle|\!\}^s_{K_{ab}}} \end{array}$$

where $\overline{a_1 \cdot a_2 \cdots a_n}$ denotes the term $\langle a_1, \langle a_2, \langle \cdots a_n \rangle \cdots \rangle$ for $a_1, \ldots, a_n \in \Sigma$. The key K_{ab} is a long-term key shared between a pair of agents a and b. The protocol describes one role. The first message simply outputs k encryptions of the pairs (u_i, v_i) that form the input of the PCP instance. The second rule offers the possibility to concatenate: when receiving an encrypted pair, the protocol replies with k encryptions, appending u_i and v_i "under the encryption".

It is easy to see that if there is a solution

$$u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots v_{i_n}$$

to the entry $(u_i, v_i)_{1 \leq i \leq k}$, then the corresponding protocol P admits an attack against the secrecy of s. Indeed the attacker can select the message $\{\!|\langle \overline{u_{i_1}}, \overline{v_{i_1}} \rangle|\!\}^s_{K_{ab}}$ in the first message from A and forward it to

B. B then replies with

$$\{|\langle \overline{u_{i_1} \cdot u_1}, \overline{v_{i_1} \cdot v_1} \rangle|\}^s_{K_{ab}}, \{|s|\}^s_{\{|\langle \overline{u_{i_1} \cdot u_1}, \overline{u_{i_1} \cdot u_1} \rangle|\}^s_{K_{ab}}}, \ldots, \{|\langle \overline{u_{i_1} \cdot u_k}, \overline{v_{i_1} \cdot v_k} \rangle|\}^s_{K_{ab}}, \{|s|\}^s_{\{|\langle \overline{u_{i_1} \cdot u_k}, \overline{u_{i_1} \cdot u_k} \rangle|\}^s_{K_{ab}}}$$

The attacker selects $\{|\langle \overline{u_{i_1} \cdot u_{i_2}}, \overline{v_{i_1} \cdot v_{i_2}} \rangle|\}^s_{K_{ab}}$ and proceeds recursively, building step by step the solution to the PCP problem. In the end, the attacker gets

$$\{|\langle \overline{u_{i_1} \cdot u_{i_2} \cdots u_{i_n}}, \overline{v_{i_1} \cdot v_{i_2} \cdots v_{i_n}} \rangle|\}^s_{K_{ab}}, \{|s|\}^s_{\{|\langle \overline{u_{i_1} \cdot u_{i_2} \cdots u_{i_n}}, \overline{u_{i_1} \cdot u_{i_2} \cdots u_{i_n}} \rangle|\}^s_{K_{ab}}}$$

Since $u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots v_{i_n}$, the attacker easily deduces s.

The converse direction is more complex to prove and requires a precise model of execution. However, an interesting point to note is that this undecidability result is independent of the specificities of a particular model.

It is also interesting to note that this construction does use nonces, which means that secrecy is undecidable even for protocols that do not make use of nonces (or only use a bounded number of nonces). If additionally the size of messages is bounded, then secrecy is decidable Durgin et al. [1999]. However, when message size is bounded, secrecy is again undecidable as soon as protocols are allowed to use an unbounded number of nonces Amadio and Charatonik [2002].

8.2 Analysis of protocols with Horn clauses

8.2.1 Horn clauses

A Horn clause is a clause with at most one positive literal. Following the tradition in logic programming we write Horn clauses, also called rules, as an implication

$$H_1 \wedge \ldots \wedge H_n \rightarrow C$$

where $H_1, \ldots, H_n$ are atomic formulas called the hypotheses and C is an atomic formula called the conclusion. Moreover, we sometimes consider the hypotheses as a set $H = \{H_1, \ldots, H_n\}$ and simply write

$H \to C$. We only consider closed formulas and all variables are implicitly universally quantified.

Atomic formulas, also called facts, are built by applying a predicate on terms. We consider a single, unary predicate, the attacker predicate, denoted $\mathsf{att}(t)$. Intuitively, when $\mathsf{att}(t)$ is true, the attacker has learnt the message modelled by the term t. The construction of the terms has one peculiarity: names are parametrized by terms. As we see below this is useful to model freshness. One may see names as a particular sort of function symbol, and we use square brackets to avoid confusing them with "normal" function symbols.

$t ::=$	term
x, y, z	variable in $\mathcal{X}$
$a[t_1, \ldots, t_n]$	name in $\mathcal{N}$
$f(t_1, \ldots, t_n)$	function application for $f \in \mathcal{F}$

We present both the attacker capabilities and the protocol itself can be modelled as Horn clauses.

8.2.2 Attacker capabilities

Starting from an inference system, it is straightforward to model the attacker capabilities as Horn clauses. Recall the Dolev Yao inference system $\mathcal{I}_{DY}$:

$$\frac{x \quad y}{\langle x, y \rangle} \qquad \frac{\langle x, y \rangle}{x} \qquad \frac{\langle x, y \rangle}{y}$$

$$\frac{x \quad y}{\mathsf{senc}(x, y)} \qquad \frac{\mathsf{senc}(x, y) \quad y}{x} \qquad \frac{x \quad y}{\mathsf{aenc}(x, y)} \qquad \frac{\mathsf{aenc}(x, \mathsf{pk}(y)) \quad y}{x}$$

This inference system is modelled by the following set of Horn clauses:

$$\begin{aligned}
\mathsf{att}(x) \wedge \mathsf{att}(y) &\to \mathsf{att}(\langle x, y \rangle) \\
\mathsf{att}(\langle x, y \rangle) &\to \mathsf{att}(x) \\
\mathsf{att}(\langle x, y \rangle) &\to \mathsf{att}(y) \\
\mathsf{att}(x) \wedge \mathsf{att}(y) &\to \mathsf{att}(\mathsf{senc}(x, y)) \\
\mathsf{att}(\mathsf{senc}(x, y)) \wedge \mathsf{att}(y) &\to \mathsf{att}(x) \\
\mathsf{att}(x) \wedge \mathsf{att}(y) &\to \mathsf{att}(\mathsf{aenc}(x, y)) \\
\mathsf{att}(\mathsf{aenc}(x, \mathsf{pk}(y))) \wedge \mathsf{att}(y) &\to \mathsf{att}(x)
\end{aligned}$$

It is easy to show that $\{t_1, \ldots, t_n\} \vdash t$ if only if $\mathsf{att}(t)$ is entailed by $\{\mathsf{att}(t_1), \ldots, \mathsf{att}(t_n)\} \vdash t$ and the set of aformentionned Horn clauses represented the attacker capabilities.

More generally an inference rule $\dfrac{t_1 \cdots t_n}{t}$ can be modelled by the Horn clause

$$\mathsf{att}(t_1) \wedge \cdots \wedge \mathsf{att}(t_n) \rightarrow \mathsf{att}(t)$$

The initial knowledge of the attacker is also modelled using Horn clauses. For each term t in his initial knowledge we simply add the clause $\mathsf{att}(t)$. We suppose in particular that the attacker always knows at least one name $a[]$, and hence include the clause $\mathsf{att}(a[])$.

8.2.3 Protocols as Horn clauses

We now illustrate on the Needham Schroeder public key protocol how a protocol can itself be modelled by a set of Horn clauses. The Needham Schroeder protocol can be represented by the three following clauses:

$$\mathsf{att}(x_{pkr}) \rightarrow \mathsf{att}(\mathsf{aenc}(\langle \mathsf{pk}(sk_a[]), n_a[x_{pkr}] \rangle, x_{pkr}))$$
$$\mathsf{att}(\mathsf{aenc}(\langle x_{pki}, x_{na} \rangle, \mathsf{pk}(sk_b[]))) \rightarrow \mathsf{att}(\mathsf{aenc}(\langle x_{na}, n_b[x_{na}, x_{pki}] \rangle, x_{pki}))$$
$$\mathsf{att}(x_{pkr}) \wedge \mathsf{att}(\mathsf{aenc}(\langle n_a[x_{pkr}], x_{nb} \rangle, x_{pki})) \rightarrow \mathsf{att}(\mathsf{aenc}(x_{nb}, x_{pkr}))$$

The first message of the initiator is modelled by the first clause. The hypothesis $\mathsf{att}(x_{pk})$ models that the attacker chooses the identity (in our modeling the public key) of the responder. The conclusion models the fact that when the initiator outputs the first protocol message then the attacker learns this message. One may note that we parametrize the nonce n_a with the variables that appear in the clause's hypothesis. The aim is to ensure some kind of freshness. Indeed when the attacker executes the protocol with different agents, the nonce n_a will be different as it will be parametrized with different public keys. One may already note that executing the protocol twice with the same responder would result in using twice the same nonce.

The second clause models the reception of the responder of the first message and his sending of the second message. One may note again that the nonce is parametrized by two variables x_{na} and x_{pki}, i.e., all variables that appear in hypothesis.

The last clause models the reception of the second message and sending of the third message by the initiator. Here one may note the repetition of the hypothesis $\mathsf{att}(x_{pkr})$ of the first clause in the last clause. There are several reasons for doing this. First, it guarantees that variables are bound correctly: as the hypotheses correspond to inputs, it is possible to reuse a variable bound in the first input in any later output. Second, it gives some guarantees that the protocol can actually reach this point, i.e. previous tests were satisfied. To better understand this point consider the following example. Suppose that a participant expects a secret s_1 (not included in the attacker's initial knowledge) and outputs a constant c. Next the protocol waits for the input of constant c and then outputs the secret nonce s_2. Normally, as s_1 can never be sent by the adversary in the first place, this protocol should never perform any output and s_2 should remain secret. This protocol can be modelled using the following Horn clauses

$$\mathsf{att}(s_1[]) \rightarrow \mathsf{att}(c)$$
$$\mathsf{att}(s_1[]) \wedge \mathsf{att}(c) \rightarrow \mathsf{att}(s_2[])$$

It should be obvious that the repetition of the hypothesis $\mathsf{att}(s_1[])$ in the second clause is needed to avoid a false attack. Otherwise, the secret nonce could indeed be trivially learned by the attacker using only the second clause.

8.2.4 Approximations

While describing the modeling of protocols in Horn clauses we have already hinted towards sources of approximations.

The first source of approximations is the freshness of names. Even though names are parametrized, this may not be sufficient to faithfully represent new names. For instance when a protocol is executed several times with exactly the same inputs, the parameters will be the same and the generated nonces will be equal, while they actually should be distinct.

Another source of approximation comes from the fact that clauses may be "used" several times, and this in any order. This is best illus-

trated by an example. Consider the following protocol

$$\begin{aligned} P \;=\; & \nu s, n_1, n_2, k. \\ & \mathsf{out}(c, \langle \mathsf{senc}(n_1, k), \mathsf{senc}(n_2, k) \rangle) \\ & \mathsf{in}(c, x).\ \mathsf{out}(c, \mathsf{sdec}(x, k)) \\ & \mathsf{in}(c, x).\ \mathsf{if}\ x = \langle n_1, n_2 \rangle\ \mathsf{then}\ \mathsf{out}(c, s) \end{aligned}$$

It is easy to see that this protocol should preserve the secrecy of s, as the protocol can at most reveal one out of the two nonces. A Horn clause modeling of this protocol would be

$$\begin{aligned} & \mathsf{att}(\langle \mathsf{senc}(n_1[], k[]), \mathsf{senc}(n_2[], k[]) \rangle) \\ & \mathsf{senc}(x, k[]) \rightarrow \mathsf{att}(x) \\ & \mathsf{senc}(x, \langle n_1[], n_2[] \rangle) \rightarrow \mathsf{att}(s[]) \end{aligned}$$

However the second rule may be used twice to learn both n_1 and n_2. Then the attacker can use the pairing rule (in the attacker capabilities) and the third protocol rule to learn the secret s. The fact that a rule may be used any number of times is however important as this is the key for modeling an unbounded number of sessions.

8.2.5 Secrecy and derivability queries

In this formalism verifying whether a term t is (weakly) secret amounts to checking whether a fact $\mathsf{att}(t)$ is *derivable* from a set of horn clauses.

Before defining derivability we introduce the notion of *subsumption*.

Definition 8.1 (subsumption)**.** We say that a clause $H_1 \rightarrow C_1$ subsumes the clause $H_2 \rightarrow C_2$, written $H_1 \rightarrow C_1 \sqsupseteq H_2 \rightarrow C_2$, if and only if there exists a substitution σ such that $C_1\sigma = C_2$ and $H_1\sigma \subset H_2$.

Intuitively, a clause which subsumes another clause is more general and requires fewer hypotheses.

Definition 8.2 (derivability)**.** Given a set of Horn clauses $\mathcal{R}$, a fact f is derivable from $\mathcal{R}$ if there exists a finite tree such that

- each node is labelled by a fact,
- the root is labelled by f

- if a node is labelled by a fact f_0 and its child nodes are labelled by $f_1, \ldots, f_n$ then there exists $R \in \mathcal{R}$ such that $\{f_1, \ldots f_n\} \rightarrow f_0 \sqsubseteq R$.

Example 8.1. We consider again the Needham Schroeder public key protocol. The Horn clauses representation of this protocol, together with the relevant attacker capabilities and initial intruder knowledge are recalled in Figure 8.1.

To show that this protocol is insecure we give the tree witnessing that an instance of $\mathsf{att}(nb[x])$ is derivable in Figure 8.2. We annotate each derivation step with the rules defined in Figure 8.1. One may note that the derivation tree mimics the man-in-the middle attack. The fact that the tree has been split in three subtrees is merely for layout purposes.

8.2.6 The analysis procedure

The analysis procedure was originally defined by Blanchet [2001]. It operates in two steps. The first step is a kind of forward search that performs resolution on the set of Horn clauses, combining some of the clauses into new, "optimized" clauses. The new clauses allow to perform several derivation steps of the original set of rules as a single step. The resulting set of clauses is guaranteed to preserve derivability with respect to the original clauses.

The second step is a backward depth-first search on the "optimized" set of clauses, whose aim it is to check whether a clause is derivable.

We now explain the two steps in more details. Our presentation of the algorithm follows the one of Blanchet [2011], where the interested reader may find additional details.

Resolution algorithm

We start by defining what is a resolution step. Resolution is a classical technique for combining rules in order to derive a new rule.

Definition 8.3 (Resolution)**.** Let $R = H \rightarrow C$ and $R' = H' \rightarrow C'$ be two clauses. If there exists $F_0 \in H'$ such that F_0 and C are unifiable

Attacker capabilities:

$$\mathsf{att}(x) \wedge \mathsf{att}(y) \rightarrow \mathsf{att}(\langle x, y\rangle) \tag{8.1}$$
$$\mathsf{att}(\langle x, y\rangle) \rightarrow \mathsf{att}(x) \tag{8.2}$$
$$\mathsf{att}(\langle x, y\rangle) \rightarrow \mathsf{att}(y) \tag{8.3}$$
$$\mathsf{att}(x) \wedge \mathsf{att}(y) \rightarrow \mathsf{att}(\mathsf{aenc}(x, y)) \tag{8.4}$$
$$\mathsf{att}(\mathsf{aenc}(x, \mathsf{pk}(y))) \wedge \mathsf{att}(y) \rightarrow \mathsf{att}(x) \tag{8.5}$$

Initial knowledge:

$$\mathsf{att}(sk_c[]) \tag{8.6}$$
$$\mathsf{att}(\mathsf{pk}(sk_a[])) \tag{8.7}$$
$$\mathsf{att}(\mathsf{pk}(sk_b[])) \tag{8.8}$$
$$\mathsf{att}(\mathsf{pk}(sk_c[])) \tag{8.9}$$
$$\mathsf{att}(a[]) \tag{8.10}$$

Protocol clauses:

$$\mathsf{att}(x_{pkr}) \rightarrow \mathsf{att}(\mathsf{aenc}(\langle \mathsf{pk}(sk_a[]), n_a[x_{pkr}]\rangle, x_{pkr})) \tag{8.11}$$
$$\mathsf{att}(\mathsf{aenc}(\langle x_{pki}, x_{na}\rangle, pk(sk_b[]))) \rightarrow \mathsf{att}(\mathsf{aenc}(\langle x_{na}, n_b[x_{na}, x_{pki}]\rangle, x_{pki})) \tag{8.12}$$
$$\mathsf{att}(x_{pkr}) \wedge \mathsf{att}(\mathsf{aenc}(\langle n_a[x_{pkr}], x_{nb}\rangle, x_{pki})) \rightarrow \mathsf{att}(\mathsf{aenc}(x_{nb}, x_{pkr})) \tag{8.13}$$

Figure 8.1: Horn clauses modeling the Needham Schroeder public key protocol

and $\sigma = \mathsf{mgu}(F_0, C)$ then the clause $R \circ_{F_0} R'$, defined as

$$(H \cup (H' \setminus \{F_0\}))\sigma \rightarrow C'\sigma$$

is the resolvent obtained by resolution of R' with R using F_0.

Example 8.2. Consider again the clauses in Figure 8.1. We have that

$$(8.6) \circ_{\mathsf{att}(x_{pkr})} (8.11) = \mathsf{att}(\mathsf{aenc}(\langle \mathsf{pk}(sk_a[]), n_a[\mathsf{pk}(sk_c)]\rangle, \mathsf{pk}(sk_c)))$$

and

$$(8.4) \circ_{\mathsf{att}(\mathsf{aenc}(\langle x_{pki}, x_{na}\rangle, pk(sk_b[])))} (8.12)$$
$$= \mathsf{att}(\langle x_{pki}, x_{na}\rangle) \wedge \mathsf{att}(\mathsf{pk}(sk_b[])) \rightarrow \mathsf{att}(\mathsf{aenc}(\langle x_{na}, n_b[x_{na}, x_{pki}]\rangle, x_{pki}))$$

$$\Pi_1 := \quad \dfrac{\dfrac{}{\mathsf{pk}(skc[])}\,(8.9)}{\mathsf{att}(\mathsf{aenc}(\langle \mathsf{pk}(sk_a[]), n_a[\mathsf{pk}(sk_c)]\rangle, \mathsf{pk}(skc[])))}\,(8.11)$$

$$\Pi_2 := \quad \dfrac{\Pi_1 \quad \dfrac{}{\mathsf{att}(skc[])}\,(8.6)}{\langle \mathsf{pk}(sk_a[]), n_a[\mathsf{pk}(sk_c)]\rangle}\,(8.5)$$

$$\Pi_3 := \quad \dfrac{\dfrac{\Pi_2 \quad \dfrac{}{\mathsf{att}(\mathsf{pk}(skb[]))}\,(8.8)}{\mathsf{att}(\mathsf{aenc}(\langle \mathsf{pk}(sk_a[]), n_a[\mathsf{pk}(sk_c)]\rangle, \mathsf{pk}(sk_b[])))}\,(8.4)}{\mathsf{att}(\mathsf{aenc}(\langle n_a[\mathsf{pk}(sk_c)], n_b[n_a[\mathsf{pk}(sk_c)], \mathsf{pk}(sk_a[])]\rangle, \mathsf{pk}(sk_a[])))}\,(8.12)$$

$$\Pi_4 := \quad \dfrac{\dfrac{\Pi_3 \quad \dfrac{}{\mathsf{pk}(skc[])}\,(8.9)}{\mathsf{att}(\mathsf{aenc}(n_b[n_a[\mathsf{pk}(sk_c)], \mathsf{pk}(sk_c[]))} \qquad \dfrac{}{\mathsf{att}(skc[])}\,(8.6)}{\mathsf{att}(n_b[n_a[\mathsf{pk}(sk_c)], \mathsf{pk}(sk_a[])])}\,(8.5)$$

Figure 8.2: Derivation tree witnessing that an instance of $\mathsf{att}(nb[x])$ is derivable

However, if resolution is applied to any possible rules, one quickly runs into non termination issues.

Example 8.3. Consider the clause

$$R = \mathsf{att}(\mathsf{senc}(x, y)) \wedge \mathsf{att}(y) \to \mathsf{att}(x)$$

introduced previously in the attacker capabilities. Already this clause on its own may be a source of non-termination. One may solve this clause with itself. For clarity, consider two renamings of this clause

$$R_1 = \mathsf{att}(\mathsf{senc}(x_1, y_1)) \wedge \mathsf{att}(y_1) \to \mathsf{att}(x_1)$$
$$R_2 = \mathsf{att}(\mathsf{senc}(x_2, y_2)) \wedge \mathsf{att}(y_2) \to \mathsf{att}(x_2)$$

Then we have that $R_1 \circ_{\mathsf{att}(\mathsf{senc}(x_2,y_2))} R_2$ results into

$$R_3 = \mathsf{att}(\mathsf{senc}(\mathsf{senc}(x_2, y_2), y_1)) \wedge \mathsf{att}(y_1) \wedge \mathsf{att}(y_2) \to \mathsf{att}(x_2)$$

Applying resolution iteratively R_1 on the resolvent results into an infinite sequence of new rules of the form

$$\begin{array}{c}\mathsf{att}(\mathsf{senc}(\mathsf{senc}(\ldots\mathsf{senc}(x_n,y_n),\ldots,y_2),y_1))\\ \wedge\mathsf{att}(y_1)\wedge\mathsf{att}(y_2)\wedge\ldots\mathsf{att}(y_n)\end{array}\quad\rightarrow\quad\mathsf{att}(x_n)$$

This example illustrates the need to guide the resolution algorithm, which can be done by the means of a selection function.

Definition 8.4. A selection function *sel* is a function from a clause $H \rightarrow C$ to a set of facts F, such that $F \subseteq H$.

Intuitively, when trying to perform resolution on a clause R we only solve facts that are in $sel(R)$. In particular, the selection function should avoid facts of the form $\mathsf{att}(x)$ when x is a variable as these facts unify with any other facts. Therefore Blanchet [2001] considers in particular *basic* selection functions. A selection function sel_0 is basic if it respects the following criteria:

$$sel_0(H \rightarrow C) = \begin{cases} \emptyset & \text{if } \forall F \in H.\ \exists x \in \mathcal{X}.\ F = \mathsf{att}(x) \\ \{F\} & \text{if } F \in H \wedge \forall x \in \mathcal{X}.\ F \neq \mathsf{att}(x) \end{cases}$$

The saturation algorithm is presented in Figure 8.3. It uses the function $\text{add}(R, \mathcal{R})$ which computes the set of clauses resulting from adding the clause R to the set of clauses $\mathcal{R}$ and eliminating any subsumed clauses. The saturation algorithm itself proceeds in three steps. First it eliminates any subsumed clauses in the initial set of clauses. Second it performs resolution until reaching a fixpoint. Resolution is however only performed between selected facts and clauses for which no hypotheses are selected. Third, it returns the set of facts that have no selected hypothesis.

To better understand the algorithm let us consider a few possible selection functions.

- Let $sel(H \rightarrow C) = H$. In that case the algorithm performs a simple forward search. Indeed the algorithm uses a clause R with no hypothesis (as they verify the condition $sel(R) = \emptyset$) to get rid of an arbitrary hypothesis of another clause. At the end the algorithm returns only clauses with no hypothesis.

$$\text{add}(R, \mathcal{R}) = \begin{cases} \mathcal{R} & \text{if } \exists R' \in \mathcal{R}, R \sqsubseteq R' \\ \{R\} \cup \{R' \in \mathcal{R} \mid R' \not\sqsubseteq R\} & \text{otherwise} \end{cases}$$

Saturate($\mathcal{R}_0$)=

1. $\mathcal{R} := \emptyset$

   For each $R \in \mathcal{R}_0$ do $\mathcal{R} := \text{add}(R, \mathcal{R}_0)$

2. Iterate until a fixed point is reached

   For each $R \in \mathcal{R}$ such that $sel(R) = \emptyset$

   For each $R' \in \mathcal{R}$ such that $F_0 \in sel(R')$ and $R \circ_{F_0} R'$ is defined

   $\mathcal{R} := add(R \circ_{F_0} R', \mathcal{R})$

3. Return $\{(H \to C) \in \mathcal{R} \mid sel(H \to C) = \emptyset\}$

Figure 8.3: First step: saturation.

- Taking the other extreme case we may define $sel(R) = \emptyset$. In that case the algorithm does not modify the set of clauses except removing subsumed clauses in the first step.
- Consider the case where the selection function is basic. As discussed above, we never select hypothesis of the form $\mathsf{att}(x)$ when applying resolution. This choice avoids many cases of non termination due to the fact that $\mathsf{att}(x)$ unifies with any fact. Moreover, at the end we only keep clauses with hypotheses of the form $\mathsf{att}(x)$ (or clauses with no hypothesis at all). These hypotheses can always be satisfied (as at least $\mathsf{att}(a[])$ is initially true). A rule having a hypothesis of a different form has either been solved during resolution (and we keep only the solved form), or it has a hypothesis that is never satisfiable and hence can never occur in a derivation tree.

Blanchet has shown that for any selection function saturation preserves derivability. This is actually an adaptation of the fact that resolution with free selection is complete [de Nivelle, 1995].

$$\text{deriv}(R, \mathcal{R}, \mathcal{R}_1) = \begin{cases} \emptyset & \text{if } \exists R' \in \mathcal{R}, R \sqsubseteq R' \\ \{R\} & \text{otherwise, if } sel(R) = \emptyset \\ \bigcup\{\text{deriv}(R' \circ_{F_0} R, \mathcal{R} \cup \{R\}, \mathcal{R}_1) \mid R' \in \mathcal{R}_1, & \\ \quad F_0 \in sel(R) \text{ and} & \\ \quad R' \circ_{F_0} R \text{ is defined}\} & \text{otherwise} \end{cases}$$

$$\text{derivable}(F, \mathcal{R}_1) = \text{deriv}(F \to F, \emptyset, \mathcal{R}_1)$$

Figure 8.4: Second step: backward depth-first search.

Lemma 8.1 ([Blanchet, 2001])**.** Let F be a closed fact. F is derivable from $\mathcal{R}$ if and only if F is derivable from saturate($\mathcal{R}$).

It is obvious that steps 1 and 2 of Saturate preserve derivability. The technical points lies in showing that it is sufficient to consider only clauses R such that $sel(R) = \emptyset$ (step 3).

Backward search

The second step of the procedure is a simply backward depth first search on the set of Horn clauses obtained from the saturation procedure. The procedure derivable($F, \mathcal{R}_1$) is described in Figure 8.4.

The result of derivable($F, \mathcal{R}_1$) is a set of clauses that provides a finite representation of the instances of F that are derivable. More precisely, for each clause $R = H \to C \in \text{derivable}(F, \mathcal{R}_1)$ we have that R is derivable from $\mathcal{R}_1$, C is an instance of F and $sel(R) = \emptyset$. Moreover, for any instance F' of F that is derivable from $\mathcal{R}_1$ there exists $H \to C \in \text{derivable}(F, \mathcal{R}_1)$ and σ such that $F' = C\sigma$ and each element of $H\sigma$ is derivable from $\mathcal{R}_1$.

The set derivable($F, \mathcal{R}_1$) is computed with the help of the auxiliary function deriv($R, \mathcal{R}, \mathcal{R}_1$). The first argument $R = H \to C$ is the current goal: C is an instance of the fact F we initially wish to derive; H are the facts needed to derive C. Initially $R = F \to F$. The second argument is the set of goals we already tried. The third argument is the set of clauses $\mathcal{R}_1$ which stays invariant. The function is evaluated recursively.

The correctness of the backward search has been proven by Blanchet through the following lemma.

Lemma 8.2 ([Blanchet, 2001])**.** Let F' be a ground instance of F. F' is derivable from $\mathcal{R}_1$ if and only if there exists a clause $H \to C \in$ derivable$(F, \mathcal{R}_1)$ and a substitution σ such that $F' = C\sigma$ and any $F'' \in H\sigma$ is derivable from $\mathcal{R}_1$.

We can now put the pieces together. If derivable$(F, \text{saturate}(\text{R})) = \emptyset$ then we have that F is not derivable from $\mathcal{R}$ as a direct consequence of the two above lemmas. Moreover, if the selection function is basic and F is ground, F is derivable from $\mathcal{R}$ if and only if derivable$(F, \text{saturate}(\text{R})) \neq \emptyset$. This follows from the two following observations: if F is ground for any $H \to C \in$ derivable$(F, \text{saturate}(\mathcal{R}))$ we have that $C = F$, and as the selection function is basic all elements of H are of the form $\mathsf{att}(x)$ for some variable x which can be derived using, e.g. $\mathsf{att}(a[])$.

8.3 Exercises

Exercise 23 (**). Show that secrecy remains undecidable even when considering only protocols with atomic keys (that is, only constants are used as keys).

Exercise 24 (*). Consider the set of clauses $\mathcal{R}_{\mathcal{I}}$ corresponding to the rules symmetric encryption.

$$\mathcal{R}_{\mathcal{I}} = \{I(x) \wedge I(y) \rightarrow I(\{x\}_y), I(\{x\}_y) \wedge I(y) \rightarrow I(x)\}$$

1. Show that $I(\{a\}_{k_2})$ is derivable from $C_1 = \mathcal{R}_{\mathcal{I}} \cup \{I(k_1), I(\{k_2\}_{k_1}), I(a)\}$ using the procedure seen is this chapter with a basic selection function.

2. Show that $I(\{a\}_{k_2})$ is not derivable from $C_2 = \mathcal{R}_{\mathcal{I}} \cup \{I(k_1), I(a)\}$. You may use the procedure seen is this chapter with a basic selection function.

9

Further readings and conclusion

The aim of our tutorial was to give an introduction to some selected topics in the area of security protocol verification. Without aiming at giving an exhaustive bibliography we conclude our tutorial with a few pointers towards other approaches and some new directions.

Other approaches for protocol verification We have seen that bounding the number of sessions is sufficient to make the problem of protocol verification decidable (for several equational theories and properties). If one is willing to additionally bound the length of messages the resulting system is finite and general purpose model checkers can be used to analyze protocols. Even though this approach does not provide strong security guarantees it has been successfully used to detect flaws in protocols, see for instance the work by Mitchell et al. [1997] using Murϕ and Lowe [1997b] using Casper and FDR.

Other approaches go in the opposite directions. Instead of simplifying the model to ensure automation, they sacrifice automation in order to keep the full adversary model. Paulson [1998] has for instance used the Isabelle theorem prover to prove protocols correct. The tamarin tool [Schmidt et al., 2012] is a more recent, and dedicated tool that al-

lows for interactive proof construction when the automatic mode fails.

It is also possible to use static analysis techniques, in particular type systems. This line of work goes back to Abadi's result on secrecy by typing [Abadi, 1997]. The use of a type system is of course incomplete, i.e., some secure protocols may not type check, but the type system dictates a discipline on how a protocol should be programmed. These ideas have been generalized in many directions and we refer the reader to Foccardi and Maffei's chapter on types for security protocols [Focardi and Maffei, 2011]. One may in particular mention the F7 type checker [Bengtson et al., 2011] which allows to verify a large variety of trace properties for protocols written in the $F^{\#}$ functional language, by using refinement types and outsourcing the verification conditions to an SMT solver.

Extending the scope Most protocol verification tools allow the verification of trace properties, such as weak secrecy and some forms of correspondance properties, but do not support the verification of equivalence properties. An exception is the ProVerif tool which offers some, albeit limited, support for equivalence properties [Blanchet et al., 2005]. This is due in particular to the fact that it verifies a more fine grained property which is too strong in many examples, e.g. when verifying unlinkability in e-Passports or privacy in voting protocols. Recently, a few dedicated tools have been developed for verifying trace equivalence for a bounded number of sessions [Tiu and Dawson, 2010, Cheval et al., 2011, Chadha et al., 2012]. However, existing tools have either limited support for equational theories or do not support non trivial else branches. These tools are still at a prototype status and not as mature as the ones for trace properties. An interesting direction for future work is to design interactive tools for proving equivalence properties for an unbounded number of sessions.

Other shortcomings of existing tools are the types of protocols they can analyze. Some protocols require to maintain a global, mutable state, i.e. a memory that can be read and modified by parallel threats. A prominent example where modeling such a state is required are security hardware modules which can be accessed by a security API, such

as the RSA PKCS#11 standard, IBM's CCA or the trusted platform module (TPM). These security tokens have been known to be vulnerable to logical attacks for some time [Longley and Rigby, 1992, Bond and Anderson, 2001] and formal analysis has shown to be a valuable tool to identify attacks and find secure configurations [Delaune et al., 2010b, Bortolozzo et al., 2010]. Apart from security APIs many other protocols need to maintain databases: key servers need to store the status of keys, in optimistic contract signing protocols a trusted party maintains the status of a contract, RFID protocols maintain the status of tags and more generally websites may need to store the current status of transactions.

Many existing tools, in particular those that allow to verify protocols for an unbounded number of sessions, do not handle well this kind of state. Limitations of an encoding of memory cells are for instance discussed explicitly in the ProVerif manual [Blanchet et al., 2013, Section 6.3.3]:

"Due to the abstractions performed by ProVerif, such a cell is treated in an approximate way: all values written in the cell are considered as a set, and when one reads the cell, ProVerif just guarantees that the obtained value is one of the written values (not necessarily the last one, and not necessarily one written before the read)."

Some works [Arapinis et al., 2011, Mödersheim, 2010, Delaune et al., 2011] have nevertheless used ingenious encodings of mutable state in Horn clauses, but these encodings have limitations. The most recent advances on verification of such stateful protocols are based on the tamarin prover [Schmidt et al., 2013, Kremer and Künnemann, 2014]. These works overcome existing limitations but require user interaction.

Another kind of protocols for which only very limited tool support exists are group protocols, e.g. group key exchange protocols, where participants need to process lists or other recursive data structures. This kind of data structures also appears in web services that have to process XML documents. Some of the first results on this topic were obtained by Paulson [1997] who used the Isabelle theorem prover to prove a recursive authentication protocol. In the case of a bounded number of sessions Truderung [2005] showed decidability of secrecy

for recursive protocols and Chridi et al. [2009] adapted the constraint solving approach to analyze protocols that manipulate unbounded lists. Recently, the ProVerif tool was also adapted to be able to analyze protocols with unbounded lists, and this for an unbounded number of sessions [Blanchet and Paiola, 2010]. Finally, the tamarin prover has also been used to analyze group key agreement protocols providing support for recursive structures, loops and equational theories for Diffie Hellman exponentiation and bilinear pairings [Kremer and Künnemann, 2014].

Towards more realistic models As for other areas of verification, security proofs discussed in this tutorial are carried out in an abstract, mathematical model. In so-called Dolev-Yao models the cryptography is indeed considered as "perfect", i.e. an intruder may only manipulate terms according to some deduction rules or equations. Moreover, nonces are unguessable. This contrasts with the computational models adopted in cryptography, see e.g. [Goldwasser and Micali, 1984], where adversaries are arbitrary probabilistic polynomial time Turing machines and security is expressed in terms of the probability of an adversary to break a security property. In such models, proofs are done in a way similar to proofs in complexity theory, by reducing the problem of attacking the protocol to the problem of breaking the underlying cryptographic primitives, e.g. encryption. While these models are arguably more precise and provide better security guarantees, the proofs are more complicated, generally done by hand and also more error-prone (the models being much more complex, writing very detailed proofs becomes quickly infeasible). In their seminal work, Abadi and Rogaway [2002] show that it is sometimes possible to prove computational soundness of Dolev-Yao models. Such soundness results guarantee that a proof in a Dolev-Yao model can be translated into a proof in the computational model, yielding strong guarantees while benefiting from tool support in the more abstract model. We refer the interested reader to a survey paper [Cortier et al., 2010] on this topic for an extensive description and bibliography. There have also been some attempts to formalize and (partially) automate proofs directly in computational models. The

main successes in that directions are the CryptoVerif prover [Blanchet, 2006] and the easycrypt tool [Barthe et al., 2011], a dedicated, interactive theorem prover.

Another challenge is to directly analyse the source code rather than working with abstract models, be it symbolic, Dolev-Yao models or computational ones. The CSure tool [Goubault-Larrecq and Parrennes, 2005] relies on dedicated abstract interpretation techniques to translate security protocol written in C to a set of Horn clauses that can then be analyzed. The ASPIER tool [Chaki and Datta, 2009] also allows to analyze C programs, combining software model checking techniques (in particular counter-example guided abstraction refinement) with symbolic protocol verification. Some works also build on general purpose program analysis tools to analyze protocols written in C [Dupressoir et al., 2011] and JAVA [Küsters et al., 2012]. Type systems, such as F7 [Bengtson et al., 2011] have also been used to analyze protocols written in functional languages. Yet another approach is to generate the protocol code from verified specifications, rather than try to directly analyze existing code, see [Pironti and Sisto, 2010, Cadé and Blanchet, 2013].

Conclusion To conclude, the formal analysis of security protocols is now a mature field that offers several powerful techniques to perform security proofs or to find flaws. Many challenges remain yet to be solved such as obtaining security proofs in more accurate models, verifying the implementations, or tackling new families of protocols such as e-voting or APIs for secure elements.

Acknowledgements

We would like to thank the reviewer for the careful reading and the helpful suggestions of improvement. We also thank the editors for their suggestion to write this tutorial and their patience and support until it came to a final version.

This work has received support from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n° 258865 and the ANR project ProSe (decision ANR 2010-VERS-004).

References

M. Abadi. Secrecy by typing in security protocols. In *Proc. 3rd International Symposium on Theoretical Aspects of Computer Software (TACS'97)*, volume 1281 of *Lecture Notes in Computer Science*, pages 611–638. Springer, 1997.

M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. In *Proc. 31st International Colloquium on Automata, Languages, and Programming (ICALP'04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 46–58. Springer, 2004.

M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 387(1-2):2–32, 2006.

M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM, 2001.

M. Abadi and C. Fournet. Private authentication. *Theoretical Computer Science*, 322(3):427–476, 2004.

M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.

M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2): 103–127, 2002.

R. Amadio and W. Charatonik. On name generation and set-based analysis in the Dolev-Yao model. In *Proc. 13th International Conference on Concurrency Theory (CONCUR'02)*, Lecture Notes in Computer Science, pages 499–514. Springer, 2002.

R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. 12th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 380–394, 2000.

R. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290 (1):695–740, 2002.

S. Anantharaman, P. Narendran, and M. Rusinowitch. Intruders with caps. In *Proc. 18th International Conference on Term Rewriting and Applications (RTA'07)*, volume 4533 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2007.

M. Arapinis, T. Chothia, E. Ritter, and M. D. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*, pages 107–121. IEEE Comp. Soc. Press, 2010.

M. Arapinis, E. Ritter, and M. D. Ryan. Statverif: Verification of stateful processes. In *Proc. 24th IEEE Computer Security Foundations Symposium (CSF'11)*, pages 33–47. IEEE Comp. Soc. Press, 2011.

A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security protocols and applications. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285, Edinburgh, Scotland, 2005. Springer.

A. Armando, R. Carbone, L. Compagna, J. Cuellar, and L. Tobarra Abad. Formal analysis of saml 2.0 web browser single sign-on: Breaking the saml-based single sign-on for google apps. In *Proc. 6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008)*, pages 1–10, 2008.

F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998. ISBN 978-0-521-45520-6.

F. Baader and W. Snyder. Unification theory. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001. ISBN 0-444-50813-9, 0-262-18223-8.

M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

G. Barthe, B. Grégoire, S. Heraud, and S. Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer, 2011.

D. Basin, C. Cremers, and S. Meier. Provably repairing the ISO/IEC 9798 standard for entity authentication. In *Proc. 1st Conference on Principles of Security and Trust (POST'12)*, volume 7215 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2012.

M. Baudet. Deciding security of protocols against off-line guessing attacks. In *Proc. 12th ACM Conference on Computer and Communications Security (CCS'05)*, pages 16–25. ACM, November 2005.

M. Baudet, V. Cortier, and S. Delaune. YAPA: A generic tool for computing intruder knowledge. *ACM Transactions on Computational Logic*, 14, 2013.

J. Bengtson, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis. Refinement types for secure implementations. *ACM Trans. Program. Lang. Syst.*, 33(2):8:1–8:45, 2011.

M. Berrima, N. Ben Rajeb, and V. Cortier. Deciding knowledge in security protocols under some e-voting theories. *Theoretical Informatics and Applications (RAIRO-ITA)*, 45:269–299, 2011.

B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. of the 14th Computer Security Foundations Workshop (CSFW'01)*. IEEE Comp. Soc. Press, 2001.

B. Blanchet. Automatic Proof of Strong Secrecy for Security Protocols. In *Proc. Symposium on Security and Privacy (SP'04)*, pages 86–100. IEEE Comp. Soc. Press, 2004.

B. Blanchet. A computationally sound mechanized prover for security protocols. In *Proc. IEEE Symposium on Security and Privacy (SP'06)*, pages 140–154. IEEE Comp. Soc. Press, 2006.

B. Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

B. Blanchet. *Formal Models and Techniques for Analyzing Security Protocols*, chapter Using Horn Clauses for Analyzing Security Protocols. Volume 5 of Cortier and Kremer [2011], 2011.

B. Blanchet and M. Paiola. Automatic verification of protocols with lists of unbounded length. In *Proc. 17th ACM Conference on Computer and Communications Security (CCS'10)*, pages 573–584. ACM, 2010.

B. Blanchet, M. Abadi, and C. Fournet. Automated Verification of Selected Equivalences for Security Protocols. In *Proc. 20th Symposium on Logic in Computer Science (LICS'05)*, pages 331–340. IEEE Comp. Soc. Press, 2005.

B. Blanchet, B. Smyth, and V. Cheval. *ProVerif 1.88: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, 2013.

F. Böhl and D. Unruh. Symbolic universal composability. In *Proc. 26rd Computer Security Foundations Symposium (CSF'13)*, pages 257–271, 2013.

M. Bond and R. Anderson. API level attacks on embedded systems. *IEEE Computer Magazine*, pages 67–75, October 2001.

M. Bortolozzo, M. Centenaro, R. Focardi, and G. Steel. Attacking and fixing PKCS#11 security tokens. In *Proc. 17th ACM Conference on Computer and Communications Security (CCS'10)*, pages 260–269. ACM, 2010.

M. Brusò, K. Chatzikokolakis, and J. den Hartog. Formal verification of privacy for rfid systems. In *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*, pages 75–88. IEEE Comp. Soc. Press, 2010.

S. Bursuc, H. Comon-Lundh, and S. Delaune. Associative-commutative deducibility constraints. In *Proc. 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS'07)*, volume 4393 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2007.

D. Cadé and B. Blanchet. Proved generation of implementations from computationally-secure protocol specifications. In *Proc. 2nd Conference on Principles of Security and Trust (POST'13)*, volume 7796 of *Lecture Notes in Computer Science*, pages 63–82. Springer, 2013.

R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symp. on Foundations of Computer Science (FOCS'01)*, pages 136–145. IEEE Comp. Soc. Press, 2001.

R. Chadha, Ş. Ciobâcă, and S. Kremer. Automated verification of equivalence properties of cryptographic protocols. In *Programming Languages and Systems —Proc. 21th European Symposium on Programming (ESOP'12)*, volume 7211 of *Lecture Notes in Computer Science*, pages 108–127. Springer, 2012.

S. Chaki and A. Datta. Aspier: An automated framework for verifying security protocol implementations. In *Proc. 22nd Computer Security Foundations Symposium (CSF'09)*, pages 172–185. IEEE Comp. Soc. Press, 2009.

V. Cheval. *Automatic verification of cryptographic protocols: privacy-type properties*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, December 2012.

V. Cheval. Apte: an algorithm for proving trace equivalence. In *Proc. 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, volume 8413 of *Lecture Notes in Computer Science*, pages 587–592. Springer, 2014.

V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: Negative tests and non-determinism. In *Proc. 18th ACM Conference on Computer and Communications Security (CCS'11)*, pages 321–330. ACM, 2011.

V. Cheval, V. Cortier, and A. Plet. Lengths may break privacy – or how to check for equivalences with length. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8043 of *Lecture Notes in Computer Science*, pages 708–723. Springer, 2013.

Y. Chevalier and M. Rusinowitch. Compiling and securing cryptographic protocols. *Inf. Process. Lett.*, 110(3):116–122, 2010.

T. Chothia and V. Smirnov. A traceability attack against e-passports. In *Proc. 14th International Conference on Financial Cryptography and Data Security (FC'10)*, volume 6052 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2010.

N. Chridi, M. Turuani, and M. Rusinowitch. Decidable analysis for a class of cryptographic group protocols with unbounded lists. In *Proc. 22nd Computer Security Foundations Symposium (CSF'09)*, pages 277–289. IEEE Comp. Soc. Press, 2009.

Ş. Ciobâcă, S. Delaune, and S. Kremer. Computing knowledge in security protocols under convergent equational theories. *Journal of Automated Reasoning*, 48(2):219–262, 2012.

H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of Exclusive Or. In *Proc. 18th Annual IEEE Symposium on Logic in Computer Science (LICS '03)*, pages 271–280, Los Alamitos, CA, 2003. IEEE Computer Society.

H. Comon-Lundh, V. Cortier, and E. Zălinescu. Deciding security properties for cryptographic protocols. Application to key cycles. *ACM Transactions on Computational Logic*, 11(2), 2010.

R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In *Proc. 9th International Static Analysis Symposium (SAS'02)*, volume 2477 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2003.

R. Corin, J. Doumen, and S. Etalle. Analysing password protocol security against off-line dictionary attacks. *ENTCS*, 121:47–63, 2005.

R. Corin, S. Etalle, and A. Saptawijaya. A logic for constraint-based security protocol analysis. In *Proc. IEEE Symposium on Security and Privacy (SP'06)*, pages 155–168. IEEE Comp. Soc. Press, 2006.

V. Cortier and S. Delaune. Decidability and combination results for two notions of knowledge in security protocols. *Journal of Automated Reasoning*, 48, 2012.

V. Cortier and S. Kremer, editors. *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*. IOS Press, 2011.

V. Cortier, S. Kremer, and B. Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *Journal of Automated Reasoning*, 46(3-4):225–259, 2010.

C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Proc. 20th International Conference on Computer Aided Verification (CAV'08)*, volume 5123 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.

M. D. Davis and E. J. Weyuker. *Computability, complexity and languages*, chapter 7, pages 128–132. Computer Science and Applied Mathematics. Academic Press, 1983.

H. de Nivelle. *Ordering Refinements of Resolution*. PhD thesis, Technische Universiteit Delft, 1995.

S. Delaune and F. Jacquemard. A decision procedure for the verification of security protocols with explicit destructors. In *Proc. 11th ACM Conference on Computer and Communications Security (CCS'04)*, pages 278–287. ACM, 2004.

S. Delaune and F. Jacquemard. Decision procedures for the security of protocols with probabilistic encryption against offline dictionary attacks. *Journal of Automated Reasoning*, 36(1-2):85–124, 2006.

S. Delaune, S. Kremer, and O. Pereira. Simulation based security in the applied pi calculus. In *Proc. 29th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'09)*, volume 4 of *Leibniz International Proceedings in Informatics*, pages 169–180. Leibniz-Zentrum für Informatik, 2009a.

S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009b.

S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi calculus. *Journal of Computer Security*, 18(2):317–377, 2010a.

S. Delaune, S. Kremer, and G. Steel. Formal analysis of PKCS#11 and proprietary extensions. *Journal of Computer Security*, 18(6):1211–1245, 2010b.

S. Delaune, S. Kremer, M. D. Ryan, and G. Steel. Formal analysis of protocols based on TPM state registers. In *Proc. 24th IEEE Computer Security Foundations Symposium (CSF'11)*, pages 66–82. IEEE Press, 2011.

S. Delaune, S. Kremer, and D. Pasaila. Security protocols, constraint systems, and group theories. In *Proc. 6th International Joint Conference on Automated Reasoning (IJCAR'12)*, volume 7364 of *Lecture Notes in Artificial Intelligence*, pages 164–178. Springer, 2012.

W. Diffie and M. Helman. New directions in cryptography. *IEEE Transactions on Information Society*, 22(6):644–654, 1976.

D. Dolev and A.C. Yao. On the security of public key protocols. In *Proc. 22nd Symposium on Foundations of Computer Science*, pages 350–357. IEEE Comp. Soc. Press, 1981.

H. Dong, N.and Jonker and J. Pang. Analysis of a receipt-free auction protocol in the applied pi calculus. In *Proc. International Workshop on Formal Aspects in Security and Trust (FAST'10)*, volume 6561 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 2011.

J. Dreier, P. Lafourcade, and Y. Lakhnech. Formal verification of e-auction protocols. In *Proc. 2nd Conferences on Principles of Security and Trust (POST'13)*, volume 7796 of *Lecture Notes in Computer Science*, pages 247–266. Springer, 2013.

F. Dupressoir, A. D. Gordon, J. Jürjens, and D. A. Naumann. Guiding a general-purpose c verifier to prove cryptographic protocols. In *Proc. 24th Computer Security Foundations Symposium (CSF'11)*, pages 3–17. IEEE Comp. Soc. Press, 2011.

N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. Workshop on Formal Methods and Security Protocols*, 1999.

S. Escobar, C. Meadows, and J. Meseguer. Maude-npa: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V*, volume 5705 of *Lecture Notes in Computer Science*, pages 1–50. Springer, 2009.

R. Focardi and M. Maffei. *Formal Models and Techniques for Analyzing Security Protocols*, chapter Types for Security Protocols. Volume 5 of Cortier and Kremer [2011], 2011.

R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *Proc. World Congress on Formal Methods (FM'99)*, Lecture Notes in Computer Science, pages 794–813. Springer, 1999.

A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology - AUSCRYPT'92*, volume 718 of *Lecture Notes in Computer Science*, pages 244–251. Springer-Verlag, 1992.

Th. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Proc. 17th International Conference on Automated Deduction (CADE'00)*, volume 1831 of *Lecture Notes in Computer Science*, pages 271–290. Springer, 2000.

S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28, 1984.

D. Gollmann. What do we mean by entity authentication? In *Proc. Symposium on Security and Privacy (SP'96)*, pages 46–54. IEEE Comp. Soc. Press, 1996.

J. Goubault-Larrecq. A method for automatic cryptographic protocol verification (extended abstract). In *Proc. Workshops of the 15th International Parallel and Distributed Processing Symposium*, volume 1800 of *Lecture Notes in Computer Science*, pages 977–984. Springer, 2000.

J. Goubault-Larrecq and F. Parrennes. Cryptographic protocol analysis on real C code. In *Proc. 6th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, volume 3385 of *Lecture Notes in Computer Science*, pages 363–379. Springer, 2005.

ISO/IEC-9798-1. *Information technology - Security techniques - Entity authentication mechanisms; Part 1: General model. ISO/IEC 9798-1.* International Organization for Standardization, second edition edition, sep 1991.

F. Jacquemard, M.Rusinowitch, and L. Vigneron. Compiling and verifying security protocols. In *Proc. 7th International Conference on Logic for Programming and Automated Reasoning (LPAR'00)*, volume 1955 of *Lecture Notes in Computer Science*, pages 131–160. Springer, 2000.

S. Kremer and R. Künnemann. Automated analysis of security protocols with global state. In *Proceedings of the 35th IEEE Symposium on Security and Privacy (SP'14)*, pages 163–178. IEEE Comp. Soc. Press, 2014.

S. Kremer and L. Mazaré. Adaptive soundness of static equivalence. In *Proc. 12th European Symposium on Research in Computer Security (ESORICS'07)*, volume 4734 of *Lecture Notes in Computer Science*, pages 610–625. Springer, 2007.

S. Kremer and M. D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In *Programming Languages and Systems — Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200. Springer, 2005.

S. Kremer, A. Mercier, and R. Treinen. Reducing equational theories for the decision of static equivalence. *Journal of Automated Reasoning*, 48(2): 197–217, 2012.

R. Küsters, T. Truderung, and J. Graf. A framework for the cryptographic verification of java-like programs. In *Proc. 25th Computer Security Foundations Symposium (CSF'12)*, pages 198–212. IEEE Comp. Soc. Press, 2012.

D. Longley and S. Rigby. An automatic search for security flaws in key management schemes. *Computers and Security*, 11(1):75–89, March 1992.

G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

G. Lowe. A hierarchy of authentication specifications. In *Proc. 10th Computer Security Foundations Workshop (CSFW'97)*, pages 31–44. IEEE Comp. Soc. Press, 1997a.

G. Lowe. Casper: a compiler for the analysis of security protocols. In *Proc. 10th Computer Security Foundations Workshop (CSFW'97)*, pages 18–30. IEEE Comp. Soc. Press, 1997b.

G. Lowe. Analysing protocols subject to guessing attacks. *Journal of Computer Security*, 12(1):83–98, 2004.

C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

J. Millen. A necessarily parallel attack. In *FMSP '99*, 1999.

J. Millen and G. Denker. Capsl and mucapsl. *J. Telecommunications and Information Technology*, 4:16–27, 2002.

J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. 8th ACM Conference on Computer and Communications Security (CCS'01)*, 2001.

J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murφ. In *Proc. IEEE Symposium on Security and Privacy (SP'97)*, pages 141–153, 1997.

S. Mödersheim. Abstraction by set-membership: verifying security protocols and web services with databases. In *Proc. 17th ACM Conference on Computer and Communications Security (CCS'10)*, pages 351–360. ACM, 2010.

D. Monniaux. Abstracting cryptographic protocols with tree automata. *Sci. Comput. Program.*, 47(2-3):177–202, 2003.

R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

L. C. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proc. 10th Computer Security Foundations Workshop (CSFW'97)*, pages 84–95. IEEE Comp. Soc. Press, 1997.

L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1/2):85–128, 1998.

Alfredo Pironti and Riccardo Sisto. Provably correct Java implementations of Spi Calculus security protocols specifications. *Computers & Security*, 29: 302–314, 2010.

M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. 14th Computer Security Foundations Workshop (CSFW'01)*, pages 174–190. IEEE Comp. Soc. Press, 2001.

M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete. *Theoretical Computer Science*, 299:451–475, 2003.

P.Y.A Ryan, S.A. Schneider, M. Goldsmith, G. Lowe, and A.W. Roscoe. *Modelling and Analysis of Security Protocols*. Addison Wesley, 2000.

B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *Proc. 25th IEEE Computer Security Foundations Symposium (CSF'12)*, pages 78–94. IEEE Comp. Soc. Press, 2012.

B. Schmidt, S. Meier, C. Cremers, and D. Basin. The tamarin prover for the symbolic analysis of security protocols. In *Proc. 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

S. Schneider. Verifying authentication protocols with CSP. In *Proc. 10th Computer Security Foundations Workshop (CSFW'97)*. IEEE Comp. Soc. Press, 1997.

D. Song. Athena, an automatic checker for security protocol analysis. In *Proc. 12th IEEE Computer Security Foundations Workshop (CSFW'99)*. IEEE Comp. Soc. Press, 1999.

J. Thayer, J. Herzog, and J. Guttman. Strand spaces: proving security protocols correct. *IEEE Journal of Computer Security*, 7:191–230, 1999.

A. Tiu and J. Dawson. Automating open bisimulation checking for the spi-calculus. In *Proc. 23rd Computer Security Foundations Symposium (CSF'10)*, pages 307–321. IEEE Comp. Soc. Press, 2010.

T. Truderung. Selecting theories and recursive protocols. In *Proc. 16th International Conference on Concurrency Theory (Concur'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 217–232. Springer, 2005.

Ch. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In *Proc. 16th International Conference on Automated Deduction (CADE'99)*, volume 1632 of *Lecture Notes in Computer Science*, pages 314–328. Springer, 1999.

T.Y.C. Woo and S.S. Lam. Authentication for distributed systems. In *Proc. Symposium on Security and Privacy (SP'92)*, pages 178–194. IEEE Comp. Soc. Press, 1992.